

KAMPOW

MICRO APPLICATION

3

AMSTRAD

LE BASIC AU BOUT DES DOIGTS
CPC 464, 664 et 6128



UN LIVRE DATA BECKER



MICRO APPLICATION

3

AMSTRAD

LE BASIC AU BOUT DES DOIGTS

CPC 464, 664 et 6128



UN LIVRE DATA BECKER

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
9, Rue Jacob
75006 PARIS

ISBN : 2-86899-006-1

Traduction Française assurée par Pascal HAUSMAN

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

TABLE DES MATIERES

INTRODUCTION

CHAPITRE 1

CHAPITRE 2

CHAPITRE 3

CHAPITRE 4

CHAPITRE 5

Annexe

INTRODUCTION

Cet ouvrage, comme son titre l'indique, est un livre d'entraînement. Il s'adresse à tous ceux qui viennent d'acheter un CPC et qui pensent qu'ils peuvent maintenant se lancer dans la programmation.

Bien entendu ce n'est pas aussi simple. Il faut un certain travail avant de pouvoir employer l'ordinateur utilement, à moins qu'on ne se contente d'utiliser les logiciels commerciaux. C'est pourquoi cet ouvrage a pour but de vous introduire de manière très systématique à la programmation. Vous apprendrez ainsi comment on transforme un problème en un programme et comment on écrit un programme de manière lisible et rationnelle. Nous ne serions donc trop vous recommander de lire cet ouvrage en ayant toujours à portée de la main votre ordinateur pour y entrer directement les exemples de programmes que nous vous fournirons.

La première partie de cet ouvrage traite des bases générales de la programmation. Comment arriver à un bon style de programmation ? Comment commenter un programme de façon claire ? Nous répondrons à ces questions et nous vous donnerons également quelques notions de base sur le traitement des données.

Les 2ème et 3ème parties seront consacrées au travail de programmation proprement dit. Nous vous montrerons tout d'abord avec de nombreux exemples comment les diverses instructions BASIC peuvent être employées et à quoi elles servent. Nos programmes d'exemple peuvent d'ailleurs tourner -à peu d'exceptions près- sur d'autres ordinateurs disposant du même jeu d'instructions. Nous avons en effet évité d'utiliser dans nos programmes trop de PEEK et de POKE car ces instructions permettent d'obtenir des résultats qui dépendent directement des adresses spécifiques de chaque ordinateur.

L'utilisation de ces instructions produit donc des effets fort différents suivant l'ordinateur employé.

Vous trouverez à la suite des différents chapitres des exercices que nous vous invitons vivement à essayer de résoudre. Ils vous permettront en effet de vérifier que vous avez bien suivi nos développements jusqu'ici. Vous trouverez bien sûr également les solutions de ces exercices largement expliquées.

La 4ème partie traite alors de problèmes plus complexes auxquels répondent aussi des programmes plus complexes. Nous essaierons de vous montrer comment maîtriser de tels problèmes et nous vous fournirons ici aussi de nombreux exemples et des exercices avec leurs solutions, de façon à ce que vous n'ayez pas seulement à lire mais que vous puissiez effectivement vous entraîner.

La 5ème partie vous introduira enfin brièvement dans le domaine des instructions musicales et graphiques de votre CPC.

Il ne nous reste plus maintenant qu'à vous souhaiter beaucoup de plaisir et de réussite dans votre travail avec cet ouvrage. Surtout ne vous désespérez pas si vous rencontrez de temps à autre une petite difficulté. Personne n'a encore réussi à s'improviser programmeur du jour au lendemain et surtout, le travail de programmation exige qu'on puisse avoir une certaine patience dans la recherche des erreurs.

1. BASES DE LA PROGRAMMATION

1.1 EXPLICATIONS GENERALES

Nous allons traiter dans ce chapitre essentiellement des bases de la programmation. Avant que nous ne montrions avec des exemples simples puis plus complexes ce qu'est la programmation avec des instructions BASIC, nous allons nous attacher ici à des notions plus générales, c'est-à-dire que nous allons expliquer comment un problème peut être transformé en un programme. Ces petits développements théoriques pourront vous sembler peu attrayants mais ils n'en sont pas moins très utiles et ils vous aideront à aborder plus tard des programmes plus complexes.

Qu'est-ce au juste que la programmation?

Vous devez partir du principe qu'un ordinateur que vous venez d'allumer est parfaitement "stupide": il possède un langage intégré mais si vous tapez simplement sur le clavier "calcule le volume d'une sphère de 2 mètres de rayon", il ne faut pas vous attendre à ce que l'ordinateur vous réponde autre chose que "syntax error". Si vous voulez en effet faire résoudre un problème de cet ordre par votre ordinateur, vous devez indiquer à celui-ci ce qu'il doit faire non pas en français mais dans son langage propre. Vous devez donc écrire dans un ordre logique la liste des tâches qu'il devra accomplir pour trouver une solution à votre problème avec les moyens dont il dispose. La marche à suivre que vous définissez de cette manière s'appelle un "algorithme". La suite d'instructions dont elle se compose est un "programme".

Le langage que vous utilisez pour communiquer avec votre CPC est le BASIC. Le BASIC a été développé en 1961 au Darthmouth College dans le New Hampshire (USA) et son nom est une abréviation de "BEGINNERS' ALL purpose SYMBOLIC INSTRUCTION CODE",ce qui signifie "code d'instructions symboliques polyvalent pour débutants".

Le BASIC a été développé à partir du langage de programmation FORTRAN. Depuis lors sont apparus d'ailleurs sur les différents ordinateurs possèdant le langage BASIC des dialectes de ce langage, de sorte que le BASIC du CPC par exemple ne peut être utilisé directement sur d'autres ordinateurs. Les différences entre les différents dialectes BASIC ne concernent le plus souvent que des points de détail mais il est important de savoir que les programmes réalisés avec la version CPC du BASIC ne pourront parfois tourner sur d'autres ordinateurs qu'après avoir subi quelques adaptations.

L'ordinateur ne comprend par ailleurs pas directement les instructions BASIC. Il faut en effet que celles-ci soient d'abord traduites dans le langage de la machine, appelé langage-machine, qui est le seul langage de programmation avec lequel l'ordinateur travaille directement. La traduction des instructions BASIC est effectuée par l'interprèteur BASIC de l'ordinateur. Si vous entrez une instruction BASIC au clavier et que vous appuyez sur la touche ENTER, cette instruction est d'abord lue par l'interprèteur, puis traduite en code en langage-machine et ensuite seulement exécutée.

Nous dirons donc pour nous résumer que la programmation consiste à traduire un algorithme dans un langage de programmation, le langage qui nous intéresse étant le BASIC.

Voici comment les débutants mais aussi beaucoup de programmeurs plus expérimentés procèdent :

Monsieur Durand souhaite faire calculer le volume d'une sphère avec 20 rayons différents. Il lit la formule dans son livre de géométrie: V=4PIr3/3 et il la "rentre" dans l'ordinateur telle quelle :

```
10 FOR I=1 TO 20
20 INPUT"QUEL RAYON (EN CM)";R
30 V=4*PI*R^3/3
40 PRINT"LE VOLUME EST DE: ";V;" cm3"
50 NEXT I
```

Le programme tourne parfaitement et monsieur Durand obtient les résultats des calculs dont il avait besoin. Que peut-il vouloir de plus ? Il a trouvé un algorithme correspondant au problème qu'il voulait résoudre et a traduit cet algorithme en BASIC. Des programmes aussi simples donnent toujours la tentation de procéder de cette façon.

Mais dès que les problèmes à résoudre deviennent plus complexes, l'ordinateur se venge car vous ne pouvez plus dès lors saisir d'un seul coup d'oeil le flux de données et le déroulement du programme. Il peut ainsi arriver qu'un programme tourne de façon incorrecte et que vous récoltiez ainsi des résultats erronnés. Cela provient alors du fait que vous avez certainement fait une erreur de logique dans votre programme et que celui-ci ne tourne pas comme vous vouliez qu'il tourne. L'être humain a en effet en général du mal à penser "à la place" de l'ordinateur et à anticiper parfaitement toutes ces réactions. Pour que l'ordinateur puisse résoudre un problème, celui-ci doit être décomposé en de nombreuses étapes élémentaires que l'ordinateur pourra suivre dans l'ordre où elles se présentent.

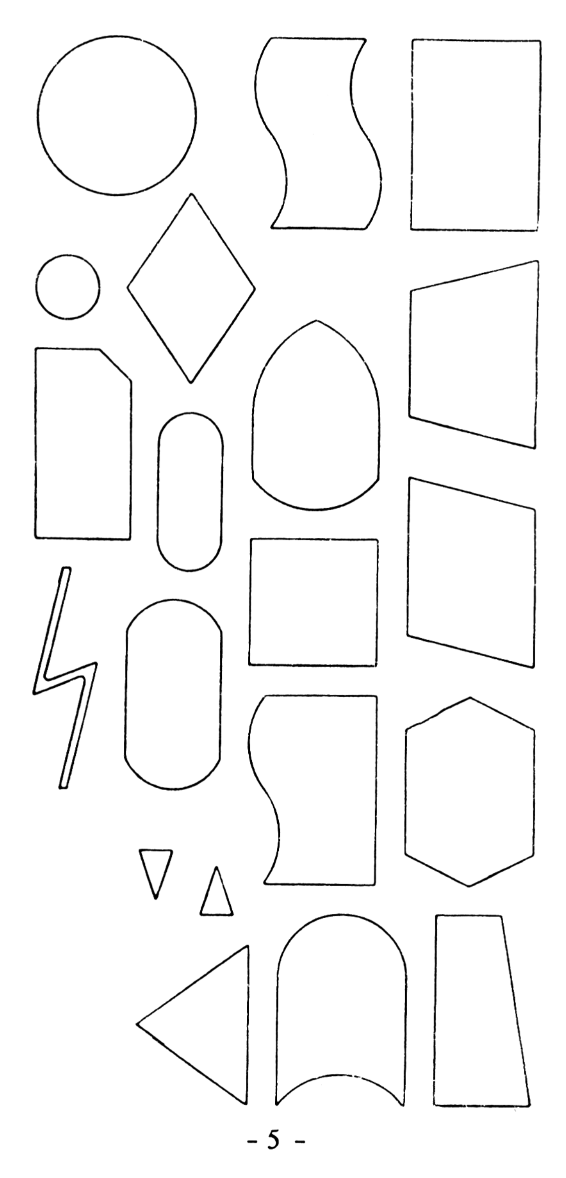
C'est justement le fait de décomposer un problème et d'ordonner correctement les diverses étapes à suivre qui provoquent toujours chez les programmeurs des erreurs plus ou moins nombreuses. Le chemin qui mène de la définition d'une tâche à un programme achevé est donc plus complexe qu'il n'y paraît au premier abord. C'est pourquoi on introduit une étape intermédiaire dans laquelle on détermine ce que l'ordinateur devra faire, et dans quel ordre il devra le faire.

1.2 FLUX DE DONNEES ET PLANS DE DEROULEMENT DU PROGRAMME

L'étape intermédiaire à laquelle nous avons fait allusion sera décrite dans cette section. Il s'agit de créer des organigrammes, flux de données et plans de déroulement du programme d'après la norme DIN 66001. Le présent chapitre constitue une brève introduction à cette technique.

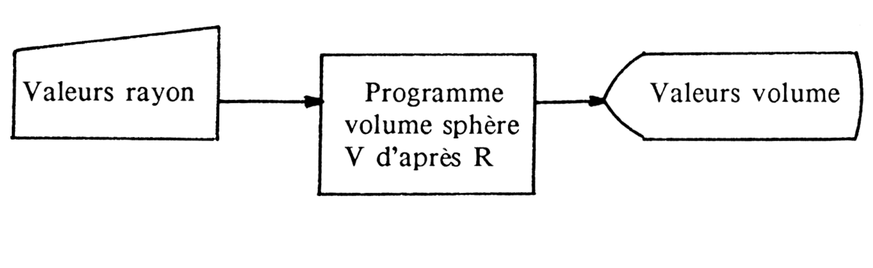
Les symboles utilisés pour réaliser des organigrammes figurent sur des plaques normographes que vous pouvez trouver dans le commerce. Voici l'illustration d'une telle plaque:

SYMBOLES POUR ORGANIGRAMMES



1.2.1 PLANS DE FLUX DE DONNEES

Les plans de flux de données doivent montrer quelles données (par exemple le rayon d'une sphère) doivent être entrées dans l'ordinateur et comment (par exemple manuellement à l'aide du clavier), par quels programmes les données seront traitées (par exemple programme de calcul du volume d'une sphère) et comment les données traitées seront ressorties (par exemple sur l'écran). Voici donc maintenant un exemple de plan de flux de données pour le programme calculant le volume d'une sphère en fonction du rayon entré manuellement:



Entrée au clavier | Traitement | Affichage à l'écran

FIGURE 2

Vous voyez qu'on peut réaliser un plan de flux de données même pour un programme aussi simple. Cela peut vous sembler ridicule mais il faut bien comprendre que la réalisation d'organigrammes sur des petits programmes n'a pas pour objet premier la réalisation de ces petits programmes mais plutôt l'apprentissage de la technique des organigrammes qui, lorsqu'elle vous sera devenue familière à force d'exercices aussi simples que celui-ci, vous rendra de grands services pour vous attaquer à des programmes beaucoup plus

complexes où la réalisation d'organigrammes devient complètement indispensable. Les organigrammes pour de gros programmes peuvent fort bien occuper plusieurs pages.

Il faut en effet reconnaître que sur les listings de très gros programmes il est extrêmement difficile de suivre le traitement des différentes données et que, le plus souvent, seul le programmeur auteur du programme y arrive encore péniblement. Si vous vous habituez dès le départ à travailler avec des plans de flux de données, vous aurez beaucoup moins de mal à en réaliser lorsque cela deviendra absolument indispensable à cause de la taille du programme.

La figure 3 explique la signification des différents symboles de plans de flux de données.

Avant de poursuivre la lecture de ce chapitre, nous vous proposons d'essayer de réaliser un plan de flux de données pour un programme convertissant les milles en kilomètres et affichant le résultat à l'écran. Comparez le plan de flux de données que vous avez réalisé avec la solution proposée par la figure 6.

Comme nous l'avons vu, les plans de flux de données permettent de représenter quelles données arriveront dans l'ordinateur à travers quels récepteurs de données, par quels programmes ces données seront traitées pour obtenir d'autres données et sur quels récepteurs de données ces nouvelles données seront sorties.

Nous allons maintenant en venir au deuxième stade de l'étape intermédiaire qui est l'objet de ce chapitre, le plan de déroulement du programme ou organigramme proprement dit. En effet nos plans de flux de données ne nous disent pas par exemple comment le rayon d'une sphère va être traité pour obtenir le volume de cette sphère.

Nous avons donc besoin d'une seconde forme de représentation symbolique décrivant toutes les étapes suivies par l'ordinateur pour résoudre un problème.

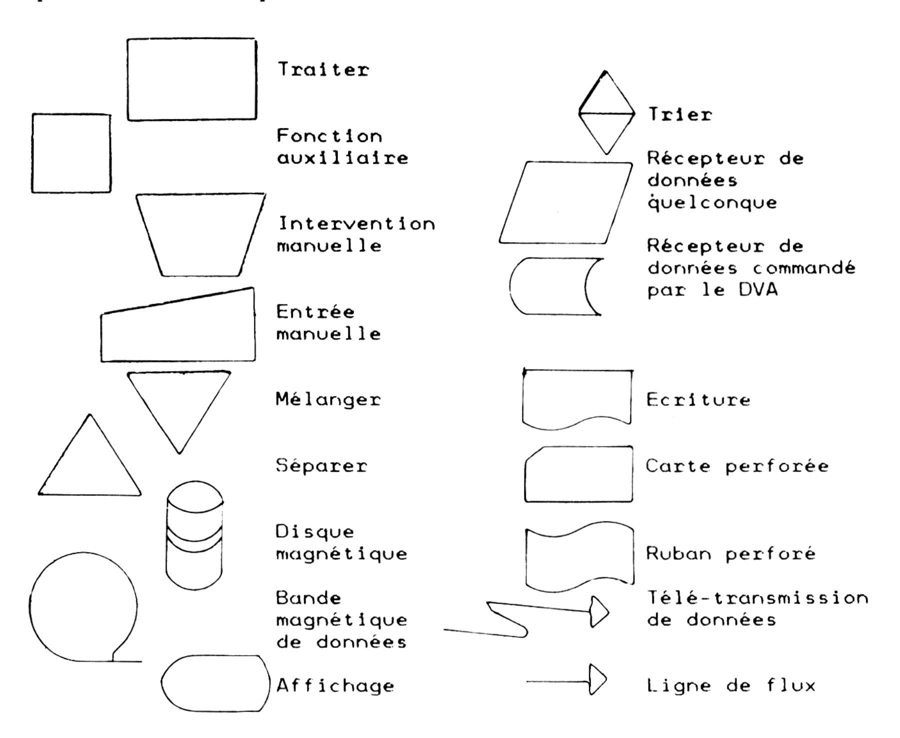


FIGURE 3

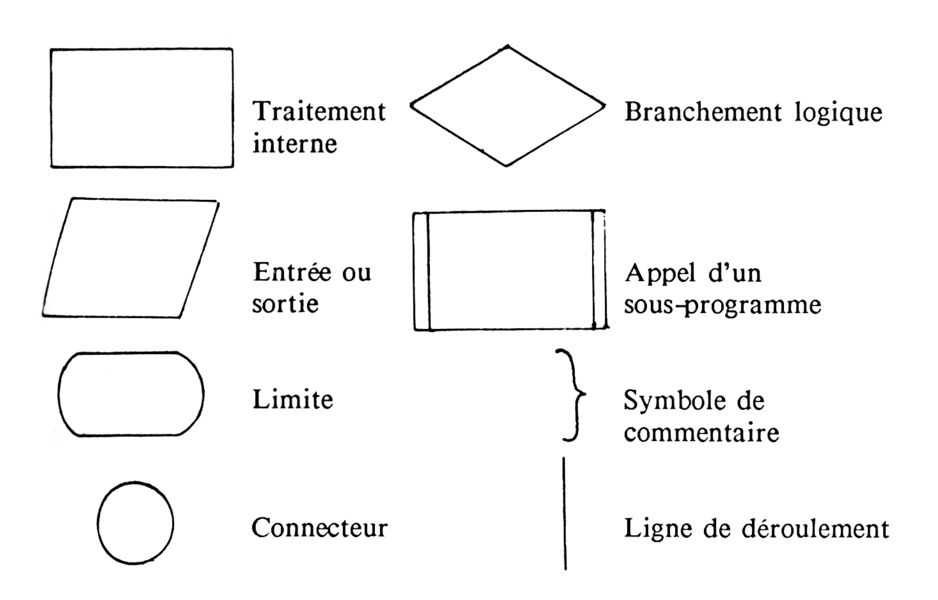

Traitement
interne
Branchement logique
Entrée ou
sortie
Appel d'un
sous-programme
Limite
Symbole de
commentaire
Connecteur
Ligne de déroulement

FIGURE 4

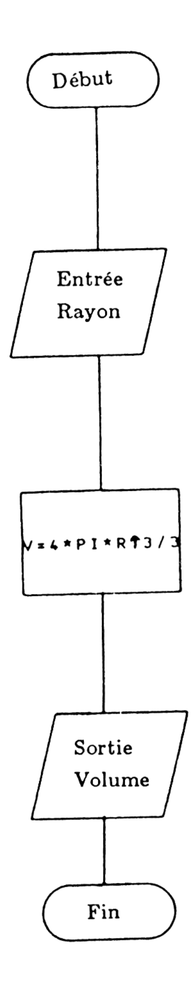


FIGURE 5

1.2.2 LES ORGANIGRAMMES

Le plan de flux de données pour le calcul du volume d'une sphère indique sous le point "traitement" uniquement "programme volume d'une sphère, V d'après R". Ceci indique donc seulement ce que deviennent les données entrées. Le problème lui-même n'a pas été décomposé en différentes étapes de traitement. C'est ce que doit réaliser l'organigramme. Un organigramme doit permettre de suivre les différentes étapes de la résolution d'un problème par un ordinateur. On utilise ici aussi les symboles conformes à la norme DIN 66001 que vous pouvez également trouver sur une plaque normographe. La signification des différents symboles est expliquée à la figure 4.

Nous allons maintenant essayer de réaliser l'organigramme correspondant toujours à notre même problème d'exemple. Comme nous l'avons déjà fait pour les plans de flux des données, nous ne saurions trop vous recommander de recourir systématiquement aux organigrammes, même pour des programmes pour lesquels cela ne s'impose pas, de façon à ce que vous maîtrisiez pleinement cette technique le jour où vous aurez à l'appliquer à des problèmes et donc des programmes beaucoup plus longs et compliqués.

Les organigrammes se dessinent toujours de haut en bas. Si vous arrivez au bas de la page, la partie suivante doit être dessinée à la droite du début de votre organigramme. Ne prenez surtout pas la mauvaise habitude de relier par une ligne deux parties d'un organigramme mais utilisez plutôt le symbole connecteur (voir la figure 4). Ce symbole doit être placé en bas de la première partie de votre organigramme et marqué d'un chiffre ou d'une lettre. Il suffit ensuite de placer un second connecteur, marqué de la même façon, au début de la deuxième partie de votre organigramme. Nous vous invitons maintenant à examiner l'organigramme que vous présente la figure 5.

Les symboles de début ou de fin n'ont pas à être traduits en BASIC. Le symbole "ENTREE RAYON" peut être traduit par l'instruction BASIC INPUT. Cette instruction peut en outre recevoir un commentaire tel que "QUEL RAYON EN CM". La formule de calcul du volume de la sphère peut être placée directement dans le symbole pour le traitement interne des données. Le symbole de sortie "sortie du volume" peut être traduit par une instruction PRINT suivie du texte correspondant. Au contraire de ce qui était le cas de notre programme précédent, nous n'avons pas utilisé ici de boucle FOR NEXT. Vous voyez que, lorsque l'organigramme a atteint un degré suffisant de précision, les différents symboles n'ont plus qu'à être remplacés par les instructions correspondantes du langage de programmation utilisé.

Une fois que vous en êtes arrivé à ce stade de la programmation, vous pouvez tenter un premier test de déroulement de votre programme. Ce test doit d'abord être effectué sur le papier, c'est-à-dire que vous devez suivre encore une fois le chemin des données sur votre plan de flux des données et contrôler le déroulement du programme tel qu'il est représenté par l'organigramme. Si tout vous semble correct, vous pouvez entrer votre programme en BASIC et le lancer avec l'instruction RUN.

Essayez maintenant de dessiner votre propre organigramme pour résoudre le problème suivant: convertir les températures exprimées en degrés Celsius en degrés Fahrenheit. La formule est: F=1.8*C+32. Nous espérons que vous trouverez la solution assez rapidement. Comparez votre organigramme à la figure 7.

C'est bien sûr seulement quand vous aborderez des programmes plus importants que vous prendrez pleinement conscience des avantages de la technique des organigrammes. Ceux-ci peuvent en effet être examinés et compris rapidement du fait de leurs caractère graphiques, alors qu'on ne peut tout de même pas en dire autant des longs listings de programmes BASIC. Un autre avantage des organigrammes qu'il convient de ne pas perdre de vue est que ceux-ci sont indépendants de l'ordinateur utilisé. Enfin, les organigrammes constituent un moyen privilégié pour documenter un programme.

Insistons quelque peu sur la documentation des programmes qui est négligée de façon réellement inexcusable par de nombreux programmeurs. Il faut bien se dire une fois pour toute qu'aucun programmeur ne peut garder en tête indéfiniment la structure logique d'un programme. Si vous ne documentez pas vos programmes, cela veut dire que pour apporter des modifications à un programme auquel vous n'avez plus touché depuis plusieurs semaines, vous risquez de devoir tout d'abord lire et relire le listing de façon à "comprendre" comment votre programme fonctionne. Ce n'est qu'après une étude longue et ennuyeuse de votre oeuvre que vous pourrez à nouveau la modifier.

Nous n'en dirons pas plus sur les plans de flux de données et les organigrammes et nous vous renvoyons si vous souhaitez approfondir cette matière à la littérature spécialisée dans ces questions.

Résumons une dernière fois les étapes que devrait normalement parcourir le travail de programmation:

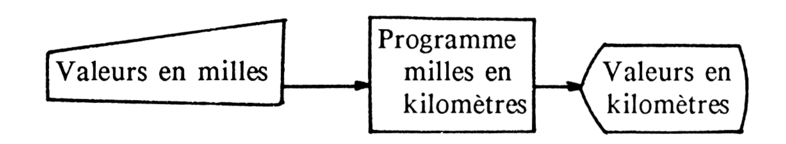
1. Définition du problème (travail sur la définition du problème, analyse du problème).

2. Projet d'algorithme de solution (plan de flux de données et organigramme ou plan de déroulement du programme).

3. Traduction de l'algorithme dans un langage de programmation (réalisation du programme).

4. Test du programme.

5. Documentation du programme.



PROPOSITION DE SOLUTION

FIGURE 6

PROPOSITION DE SOLUTION

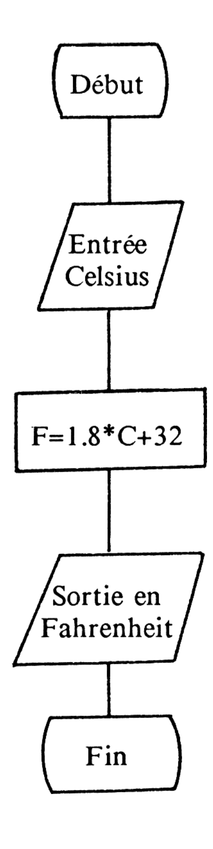


FIGURE 7

1.3 CODES ASCII

Le CPC ne peut pas traiter directement les caractères (les lettres, chiffres et autres signes) que vous entrez avec le clavier. Ces caractères ou plutôt les touches sur lesquelles vous appuyez doivent être traduits dans un code numérique. Le code numérique le plus répandu est le code ASCII. ASCII signifie "American Standard Code for Information Interchange" ce qui signifie "Code standard américain pour l'échange d'informations". Ce standard a été créé de façon à rendre possible des échanges d'informations même entre des ordinateurs différents. Par exemple c'est le caractère "A" correspondra toujours à la valeur 65 sur les ordinateurs ayant adopté le standard ASCII. Le standard ASCII utilise des valeurs allant de 0 à 127.

Le code standard ASCII réserve les valeurs 0 à 31 pour les caractères de commande, les valeurs 32-90 pour les majuscules et les valeurs 91-127 pour les minuscules et quelques autres signes. Le code ASCII du CPC correspond à ce standard. Les valeurs 128 à 255 correspondent en outre sur le CPC à des caractères graphiques particuliers. Vous pouvez vous reporter pour plus de précision aux tableaux du manuel de votre ordinateur.

1.4 LES SYSTEMES NUMERIQUES

L'ordinateur ne peut distinguer que deux états dans ces circuits électroniques: allumé et éteint. Il a donc fallu créer un système numérique adapté à cette contrainte technique, le système binaire. Le système binaire ne comporte que 2 chiffres 0 et 1, alors que le système décimal qui nous est le plus familier en comporte 10. Le 1 du système binaire correspond à l'état "allumé" des circuits électroniques et le 0 à l'état "éteint".

Pour expliquer le système binaire, partons du système décimal pour essayer de dégager les caractéristiques générales des systèmes numériques qui s'appliquent donc aussi bien au système binaire qu'au système décimal. En système décimal, nous pourrions représenter le nombre 5768 également sous la forme suivante:

5768 = 5*1000 + 7*100 + 6*10 + 8*1

ou encore:

5768 = 5*10↑3 + 7*10↑2 + 6*10↑1 + 8*10↑0

Souvenez-vous qu'en mathématiques, tout nombre élevé à la puissance 0 égale 1.

Dans le systéme décimal, les nombres sont donc représentés par une somme de produits en base 10. Chaque chiffre régit une puissance de 10 donnée:

10↑3	10↑2	10↑1	10↑0
5	6	7	8

1.4.1 LE SYSTEME BINAIRE

Le système binaire répond au même principe mais sa base est 2 au lieu de 10. C'est pourquoi ce système ne connait que deux chiffres, 0 et 1. Pour convertir en un nombre décimal le nombre binaire 1011, il nous faut procéder ainsi:

Chaque chiffre d'un nombre binaire régit une puissance de 2. Pour convertir un nombre binaire, il suffit donc d'écrire chaque chiffre en-dessous de la puissance de 2 correspondante. Il ne reste plus qu'à additionner entre elles les valeurs ainsi obtenues pour avoir le nombre décimal équivalent.

2↑3	2↑2	2↑1	2↑0
1	0	1	1

Additionnons ces valeurs :

1*2↑3 + 0*2↑2 + 1*2↑1 + 1

1*8 + 0*4 + 1*2 + 1*1 = 11

Le résultat est le nombre décimal 11. Pour convertir un nombre décimal en un nombre binaire, il faut procéder ainsi :

Pour convertir par exemple le nombre décimal 167 en un nombre binaire, il faut tout d'abord rechercher quelle est la plus grande puissance de 2 inférieure ou égale à ce nombre. Dans notre exemple, 2↑8 (=256) est supérieur à 167 mais 2↑7 (=128) est encore inférieur à 167. Cette valeur doit maintenant être soustraite du nombre décimal à convertir. Il faut ensuite procéder de même avec le reste 39. 2↑6 (=64) est supérieur à 39 mais 2↑5 (=32) est inférieur à 39.

Reste 7. 2↑4 (=16) et 2↑3 (=8) sont supérieurs à 7 mais 2↑2 (=4) est inférieur à 7. Reste 3. 2^1 (=2) est inférieur à 3. Reste 1. 2↑0 est égal à 1.
Reste 0. Nous avons maintenant relevé toutes les puissances de 2 qui composent notre nombre. Il faut simplement écrire un 1 sous ces puissances de 2 et un 0 sous les autres:

2↑7	2↑6	2↑5	2↑4	2↑3	2↑2	2↑1	2↑0
1	0	1	0	0	1	1	1

Bien sûr, si vous faites à nouveau la somme des puissances de 2 sous lesquelles figure un 0, vous retrouverez le nombre décimal 167 dont nous sommes parti.

1.4.2 BIT ET OCTET

La plus petite unité d'information que traite un ordinateur est appelée bit (BInary digiT). Un bit peut avoir deux états et peut donc représenter deux valeurs: 0 et 1. On parle de bit mis pour la valeur 1 et de bit non mis ou nul pour la valeur zéro.

Le CPC possède un microprocesseur 8 bits, c'est-à-dire qu'il ne peut placer dans une case de la mémoire que des valeurs inférieures à 256, c'est-à-dire des valeurs qui puissent être représentées avec 8 chiffres binaires, comme l'exemple précédent du nombre 167. Chaque chiffre binaire correspond donc à un bit et 8 bits constituent un octet. Si les 8 bits sont mis, nous obtenons la valeur maximum d'un octet, 255. Un octet peut donc recevoir 256 valeurs différentes de 0 à 255. Le CPC possède cependant 65536 cases de mémoire. Comment peut-il atteindre l'ensemble de ces cases, s'il ne peut placer dans une case qu'une valeur inférieure ou égale à 255 ?

Le microprocesseur de votre ordinateur ne peut atteindre ce nombre de cases mémoire qu'en scindant en deux les adresses supérieures à 255. Ces deux morceaux d'une valeur de 16 bits sont appelés low byte et high byte ou octet faible et octet fort. Les deux octets constitutifs d'une valeur 16 bits sont donc placés dans deux cases différentes de la mémoire. Pour calculer l'octet fort d'une adresse, il faut diviser cette adresse par 256. L'octet faible est le reste de cette division. Prenons un exemple:

Vous voulez placer en mémoire l'adresse 53280. Il faut la scinder en deux octets, l'octet faible et l'octet fort de cette adresse. Divisez 53280 par 256 ; vous obtenez 208 et il reste 32. La valeur de l'octet fort sera donc 208 et celle de l'octet faible 32. C'est uniquement de cette façon que l'ordinateur peut stocker les valeurs supérieures à 255. Notez bien que l'ordinateur place toujours l'octet faible avant l'octet fort. Si vous utilisez une adresse de la mémoire dans un programme, par exemple en liaison avec l'instruction POKE, cette adresse sera d'abord scindée en 2 octets, octet faible et octet fort par l'ordinateur. Les valeurs supérieures à 255 ne peuvent donc être représentées dans la mémoire de l'ordinateur que par au moins deux octets.

Cette méthode de représentation et de traitement internes des nombres rend nécessaire encore un autre système numérique, le système hexadécimal.

1.4.3 LE SYSTEME HEXADECIMAL

Le système hexadécimal repose sur la base 16. Il faut donc 16 chiffres pour représenter les nombres de ce système. On prend bien sûr les chiffres du système décimal pour représenter les valeurs de 0 à 9 mais on prend les 6 premières lettres de l'alphabet pour représenter les valeurs 10 à 15. La suite de nombres décimaux suivante:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 etc...

devient en hexadécimal:

1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 etc...

Voici maintenant quelques exemples pour nous familiariser avec le maniement des nombres hexadécimaux. Convertissons tout d'abord les nombres hexadécimaux en nombre décimaux:

2 E 0 C = 2*16^3 + 14*16^2 + 0*16^1 + 12*16^0

= 2*4096 + 14*256 + 0*16 + 12*1 = 11788

Vous voyez que nous suivons toujours le même principe pour convertir un nombre d'un système numérique dans un autre. Un autre exemple:

0 A B C = 0*16^3 + 10*16^2 + 11*16^1 + 12*16^0

= 0*4096 + 10*256 + 11*16 + 12*1 = 2748

L'intérêt principal du système hexadécimal vient du fait que la base de ce système, 16, est une puissance de 2 et se rapproche donc plus que le système décimal du système numérique binaire directement compris et utilisé par l'ordinateur. C'est pourquoi on peut distinguer presque automatiquement les octets faibles et forts qui composent un nombre hexadécimal.

Considérons l'exemple précédent 0ABC. Les deux premiers chiffres correspondent à l'octet fort. En effet un octet peut recevoir toute valeur inférieure ou égale à 255, ce qui correspond au nombre FF en hexadécimal. 0A correspond au nombre décimal 10. L'octet fort de 0ABC vaut donc 10. L'octet faible est BC, ce qui correspond à 188 (11*16 + 12) en décimal.

Vous voyez que le calcul des deux octets composant une valeur 16 bits est très facilité puisque vous pouvez distinguer aisément l'octet fort et l'octet faible et que le calcul de la valeur d'un octet exprimé en hexadécimal n'est pas très compliqué : il suffit de multiplier le chiffre de gauche par 16 et d'ajouter le résultat au chiffre de droite. La conversion des nombres binaires est également facilitée si l'on passe par le "détour" que constitue le système numérique hexadécimal. Voici quelques exemples :

Binaire	Hexadécimal	Décimal
0101 1011 =	5B	= 5*16^1 + 11*16^0 = 91
1100 0011 =	C3	= 12*16^1 + 3*16^0 = 195
1010 1010 =	AA	= 10*16^1 + 10*16^0 = 170

Vous avez certainement remarqué que nous avons divisé les nombres binaires en deux moitiés de 4 chiffres chacune. Chaque moitié a été ensuite convertie en un nombre hexadécimal. Dans le premier exemple, le premier et le troisième bits (en partant de la droite) de la moitié de gauche étaient mis, ce qui donne 5 en hexadécimal. Dans la moitié de droite, les premier, second et quatrième bits étaient mis, ce qui donne B en hexadécimal.

C'est ainsi que nous obtenons la valeur hexadécimale 5B. Chaque groupe de 4 chiffres binaires peut en effet représenter les mêmes valeurs (de 0000 à 1111 = de 0 à 15) qu'un chiffre du système hexadécimal (de 0 à F). Il est ensuite aisé de convertir le nombre hexadécimal obtenu en un nombre décimal.

Un groupe de quatre bits, qui peut donc être représenté soit par quatre chiffres binaires, soit par un chiffre hexadécimal, s'appelle un quartet.

A travers ces exemples vous voyez comment il faut procéder pour convertir des nombres d'un système numérique dans un autre. Voici maintenant en conclusion comment convertir les nombres décimaux en nombres hexadécimaux. Le principe à suivre est le même que pour la conversion des nombres décimaux en nombres binaires. Pour convertir par exemple le nombre décimal 49153 en son équivalent hexadécimal, il faut rechercher la plus grande puissance de 16 inférieure à 49153: 16^4 (=65536) est supérieur à 49153 mais 16^3 (=4096) est encore inférieur à 49153. Divisez maintenant 49153 par 16^3, vous obtenez 12, il reste 1. Nous avons presque fini: 16^2 (=256) et 16^1 (=16) sont supérieurs au reste 1 mais 16^0 (=1) est égal à 1. Le reste 1 divisé par 16^0 donne 1, reste 0:

49153 = 12*16^3 + 0*16^2 + 0*16^1 + 1*16^0

-le nombre décimal 12 correspond au chiffre hexadécimal C
-le nombre décimal 0 correspond au chiffre hexadécimal 0
-le nombre décimal 0 correspond au chiffre hexadécimal 0
-le nombre décimal 1 correspond au chiffre hexadécimal 1

Nous avons ainsi obtenu l'équivalent de 49153 en hexadécimal:

C001

Il serait peut-être temps de mettre en pratique ces notions théoriques très arides. Essayez donc de résoudre les petits problèmes de la page suivante. Si vous hésitez sur la marche à suivre, consultez à nouveau le chapitre correspondant. Les solutions se trouvent à la page suivant les exercices mais il vaut mieux ne les consulter que quand vous vous serez essayés sérieusement à résoudre ces petits exercices. Nous vous souhaitons bonne chance !

EXERCICES

1) Convertissez les nombres binaires suivants en nombres hexadécimaux:

   a) 01101100 b) 10010010
   c) 10111010 d) 11110000
   e) 00001100 f) 11001001

2) Convertissez les nombres hexadécimaux suivants en nombres décimaux:

   a) F0CA b) 1268
   c) 35A0 d) 0255
   e) F000 f) 0800

3) Convertissez les nombres binaires suivants en nombres décimaux:

   a) 10110111 b) 00110011
   c) 11111110 d) 00010101
   e) 01010101 f) 10101010

4) Convertissez les nombres décimaux suivants en nombres hexadécimaux:

   a) 63280 b) 24576
   c) 32769 d) 43981
   e) 65534 f) 18193

SOLUTIONS

1)

a) 6C	b) 92
c) BA	d) F0
e) 0C	f) C9

2)

a) 61642	b) 4712
c) 13728	d) 597
e) 61440	f) 2048

3)

a) 183	b) 51
c) 254	d) 21
e) 85	f) 170

4)

a) F730	b) 6000
c) 8001	d) ABCD
e) FFFE	f) 4711

2. INTRODUCTION A LA PROGRAMMATION

Ce chapitre a pour but de vous apprendre l'utilisation des instructions BASIC au moyen de petits programmes d'exemple. Nous suivrons les cinq étapes du travail de programmation que nous avons décrites uniquement dans le premier programme. Nous nous concentrerons ensuite sur le troisième point de ce travail, la traduction de l'algorithme en BASIC.

2.1 LE PREMIER PROGRAMME BASIC

Supposons maintenant que monsieur Durand ne veuille plus calculer le volume d'une sphère mais la surface de cette sphère pour 10 rayons différents. Il va suivre cette fois nos indications à la lettre et il commence donc par faire l'analyse du problème :

1. Définition du problème

Il veut que son CPC à partir des rayons qui lui seront fournis en centimètres, calcule la superficie S d'une sphère. La formule est :

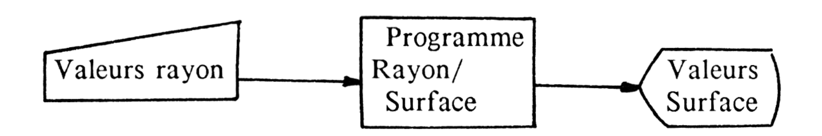
S=4PIr^2

2. Projet d'algorithme de solution

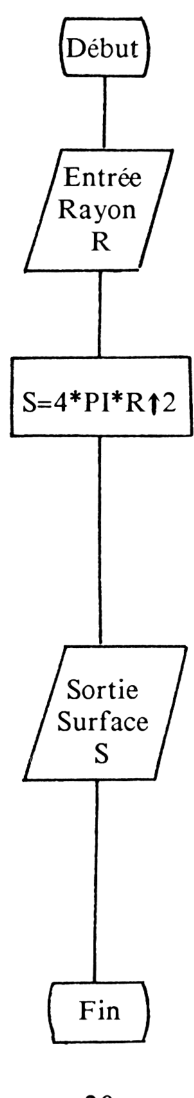
1. Début
2. Entrée de r
3. Calcul de S = 4PIr^2
4. Sortie de S sur l'écran
5. Fin

Cet organigramme est un organigramme linéaire car il ne prévoit pas de branchements sous forme de sous-programmes ou de boucles. Si vous ne comprenez pas pour le moment le sens des concepts de sous-programme et de boucle, ne vous inquiétez pas, nous les expliquerons au chapitre suivant.

PLAN DE FLUX DES DONNEES



Organigramme



3. Réalisation du programme

```
10 INPUT"QUEL RAYON EN CM";R
20 LET S=4*PI*R^2
30 PRINT S
40 END
```

4. Test du programme

Il faudrait normalement vérifier d'abord notre plan de flux de données et notre organigramme. Ceci est cependant vite fait, vu la taille très réduite de notre programme. Vous pouvez alors lancer le programme avec l'instruction RUN.

5. Documentation

Le but de la documentation est de permettre à n'importe quel autre programmeur de pouvoir comprendre et donc modifier utilement le programme. Pour un programme aussi petit que le nôtre, le plan de flux de données et l'organigramme ainsi que la courte description du programme fournie au point 1 suffisent amplement.

Vous remarquez en examinant le listing de notre programme (point 3) que chaque instruction occupe une ligne de programme. Il vaut mieux en effet éviter d'écrire des lignes pleines de nombreuses instructions car vous risquez alors de ne plus pouvoir facilement comprendre votre programme et vous risquez également d'avoir du mal à le modifier si nécessaire.

Il est également très recommandé de numéroter les lignes de vos programmes de 10 en 10 comme nous l'avons fait de façon à faciliter l'ajout éventuel de lignes supplémentaires entre deux lignes existantes.

Vous pouvez bien sûr programmer d'abord la ligne 20 puis ensuite seulement la ligne 10 puisque l'ordinateur classe automatiquement et exécute les lignes de programme dans l'ordre croissant des numéros de ligne.

Examinons maintenant les instructions employées dans notre programme: INPUT, LET, PRINT et END.

2.1.1 ENTREE DE VALEURS AVEC INPUT

L'instruction INPUT permet dans un programme l'entrée de valeurs pendant le déroulement du programme. L'utilisateur du programme (et non pas seulement le programmeur) peut donc exercer une influence directe sur le déroulement du programme. L'instruction INPUT peut être suivie d'un message placé entre guillemets comme c'est le cas dans notre programme d'exemple. Ce message doit être suivi d'une ou plusieurs variables. La première variable doit être séparée du message par un point-virgule et chaque variable doit être séparée de la suivante par une virgule. Lorsque le programme rencontre l'instruction INPUT, le curseur apparaît à l'écran et le programme est interrompu jusqu'à ce que vous entriez une entrée avec les touches du clavier, en marquant la fin de cette entrée en appuyant sur la touche ENTER. Voici quelques exemples pour plus de clarté :

a) INPUT S   Entrée: 1

La valeur 1 sera ici affectée à la variable S.

b) INPUT"QUEL RAYON";S Entrée: 3

Le texte QUEL RAYON s'affiche à l'écran et après que vous ayez appuyé sur les touches 3 et ENTER, la valeur 3 sera affectée à la variable S.

c) INPUT A,B,C Entrée: 4.3,.5,4

Les valeurs 4.3, 0.5 et 4 seront respectivement affectées au variables A, B et C. Souvenez-vous que les décimales sont placées après un point et non après une virgule comme dans la notation arithmétique française normale. N'oubliez pas que les virgules servent toujours à séparer entre elles différentes variables.

D'autre part, si vous placez une virgule entre le message suivant l'instruction INPUT et la première variable, le point d'interrogation placé automatiquement à la suite d'un message INPUT n'apparaîtra pas.

A RETENIR:

Avec INPUT les différentes variables sont séparées par des VIRGULES. Les décimales sont placées après un POINT.

L'instruction LINE INPUT est une variante de l'instruction INPUT. L'instruction LINE INPUT permet d'entrer une virgule comme partie de la variable entrée. Ceci entraîne qu'on ne peut affecter de valeur qu'à une seule variable avec LINE INPUT. Dans l'exemple donné plus haut, nous pourrions écrire également:

LINE INPUT"QUEL RAYON";S

A RETENIR:

L'instruction LINE INPUT ne permet d'affecter de valeur qu'à une seule variable. Le texte à affecter à une variable alphanumérique (chaîne de caractères) peut comprendre une virgule (voir plus loin).

2.1.2 AFFECTATION D'UNE VALEUR AVEC LET

L'instruction LET permet d'affecter une valeur à une variable. L'expression qui se trouve à la droite du signe égale est d'abord calculée par l'ordinateur puis la variable dont le nom figure à droite du signe égale reçoit la valeur de cette expression. Le mot LET n'est cependant plus utilisé sur les ordinateurs comme le CPC qui comprennent aussi bien le signe égale utilisé seul. Il faut cependant savoir que certains dialectes BASIC exigent l'emploi du mot LET.

Quelques exemples de cette instruction:

a) LET A=10 ou A=10

affecte la valeur 10 à la variable A.

b) LET A=A+5 ou A=A+5

affecte à la variable A l'ancienne valeur de A plus 5.

c) LET A=A*B-8 ou A=A*B-8

L'ancienne valeur de la variable A est multipliée par la valeur de la variable B. On retranche 8 de ce produit et le résultat est la nouvelle valeur affectée à la variable A.

L'expression à la droite du signe égale peut comprendre autant de signes mathématiques et de variables ou constantes que vous voulez. On peut donc utiliser des formules mathématiques jusqu'à un certain degré de complexité (voyez notre programme pour calculer la superficie d'une sphère).

Il ne doit ~~jamais~~ y avoir qu'une seule variable à la gauche du signe égale.

A RETENIR:

LET affecte une valeur à une variable. Il ne doit y avoir qu'une seule variable à la gauche du signe égale. A droite du signe égale peut figurer n'importe quelle expression mathématique.

Examinons à nouveau l'expression de l'exemple b). A=A+5 est absurde en mathématiques, il s'agit d'une équation fausse. Mais cette expression n'est pas une équation en BASIC mais une instruction, une affectation de valeur. Il faut considérer que les variables sont des sortes de tiroir. A=A+5 signifie qu'il faut ajouter 5 dans le tiroir A. Si ce tiroir A contenait déjà 4, il contiendra 9 une fois que l'instruction aura été exécutée. Vu sous cet angle, c'est beaucoup plus simple, non ?

2.1.3 SORTIE AVEC PRINT

L'instruction PRINT est certainement l'une des toutes premières instructions utilisées par les débutants programmeurs. C'est toutefois une des instructions les plus puissantes du BASIC de votre CPC. Il vous permet de sortir des textes, des messages ou les valeurs de diverses variables. Vous pouvez également mélanger les valeurs de variables avec des textes d'explication de la signification de ces variables.

Vous pouvez également réaliser avec l'instruction PRINT des dessins simples à l'aide des caractères graphiques de votre ordinateur. Voici quelques exemples de l'utilisation de l'instruction PRINT. Les variables utilisées auront les valeurs suivantes:

A=10 : B=20 : C=30

EXEMPLES:

	Instruction	Sortie
a)	PRINT A	10
b)	PRINT "A"	A
c)	PRINT A*B	200
d)	PRINT A,B	10 20
e)	PRINT B;C	20 30
f)	PRINT "B";B	B 20
g)	PRINT "A VAUT";A	A VAUT 10

Nous arrêterons là les exemples d'utilisation de l'instruction PRINT pour le moment, mais nous y reviendrons souvent dans nos programmes d'exemple.

D'autre part le CPC connait deux modes d'exécution des instructions: le mode direct et le mode programme. Le mode direct s'écrit comme les exemples que nous vous donnons ci-dessus. Si vous entrez donc PRINT A et que vous appuyez ensuite sur la touche ENTER, l'instruction PRINT A sera exécutée immédiatement. Le mode programme se distingue du mode direct par le fait que chaque instruction est écrite sur une ligne commençant obligatoirement par un numéro de ligne. Si vous entrez par exemple:

```
10 PRINT A
```

et que vous appuyez ensuite sur la touche ENTER, cette ligne sera placée dans la mémoire BASIC de l'ordinateur. L'instruction PRINT A ne sera cette fois exécutée que lorsque, après que vous aurez lancé le programme en mémoire en entrant RUN puis en appuyant sur la touche ENTER, le programme arrivera à la ligne 10.

Venons-en maintenant à l'exemple a). Cette façon d'écrire permet de faire afficher à l'écran la valeur d'une variable. Un 10 apparaît sur l'écran puisque nous avions auparavant affecté 10 à la variable A.

Il faut noter en ce qui concerne la sortie des nombres avec PRINT que la valeur absolue d'un nombre est toujours précédée soit d'un espace s'il s'agit d'un nombre positif, soit d'un signe moins s'il s'agit d'un nombre négatif. De cette façon les nombres 10 et -10 ont toujours la même longueur lorsqu'ils sont sortis avec PRINT.

Dans l'exemple a), comme aucun caractère ne suit le A, l'ordinateur effectue automatiquement après avoir affiché la valeur de A un retour de chariot (CARRIAGE RETURN) et un passage à la ligne (LINE FEED). Ceci signifie que la prochaine sortie avec PRINT commencera au début de la ligne suivante.

Dans l'exemple b), le A a été placé entre guillemets. Ceci entraîne que le A sera considéré comme du texte et non pas comme le nom d'une variable. C'est donc le caractère A qui sera sorti et non la valeur de la variable A. Tous les caractères placés entre guillemets sont sortis, sauf s'il s'agit de caractères de commande, par exemple pour le BIP.

L'exemple c) vous montre que vous pouvez également faire effectuer des calculs avec l'instruction PRINT. Le produit des variables A et B est ici calculé puis sorti sur l'écran. Suivent ici aussi un retour de chariot et un passage à la ligne.

L'exemple d) vous montre comment faire imprimer les valeurs de différentes variables sur une seule ligne. La virgule empêche donc ici le retour de chariot et le passage à la ligne.

Le CPC dispose en MODE 1 de 40 caractères par ligne de l'écran. Une ligne de programme peut contenir 255 caractères au maximum ce qui correspond à environ 6 lignes 1/4 en MODE 1. Si vous utilisez le point d'interrogation comme signe d'abréviation de l'instruction PRINT et que vous utilisez pleinement les 255 caractères autorisés, l'instruction PRINT sera écrite in extenso lorsque vous listerez cette ligne. Votre ligne de programme dépassera alors la limite autorisée de 255 caractères et les caractères en trop seront supprimés en fin de ligne. Il convient donc de tenir compte de cette particularité qui peut aboutir à des résultats désagréables.

Chaque ligne de l'écran du CPC est divisée à la mise sous tension en zones de 13 caractères. Ceci constitue donc une sorte de tabulation. Si vous placez une virgule entre deux variables, la seconde variable sera donc placée au début de la zone de 13 caractères suivante donc en colonne 14. La troisième variable sera placée en colonne 27, et ainsi de suite.

Si vous entrez:

PRINT "1","2","3","4","5","6","7","8"

Si vous appuyez maintenant sur la touche ENTER, vous pourrez voir l'emplacement des diverses tabulations de l'écran. Si nous n'avions pas mis les chiffres entre guillemets, ils auraient été décalés d'une case sur la droite à cause du caractère réservé pour le signe des nombres.

L'exemple e) illustre l'emploi du point-virgule. Le point-virgule interdit le retour de chariot et le passage à la ligne mais également la tabulation. Les expressions sont donc sorties à la suite l'une de l'autre, exactement comme elles ont été écrites, sans espace intercalaire. Ceci permet d'afficher un texte directement à la suite de la valeur d'une variable.

L'exemple f) vous montre comment, toujours grâce au point-virgule, vous pouvez faire afficher la valeur d'une variable directement à la suite d'un texte.

L'exemple g) utilise la même possibilité.

L'instruction END dans la dernière ligne de notre programme marque la fin logique du programme. Cette instruction n'est pas toujours placée dans la dernière ligne du programme car elle peut être suivie de sous-programmes. Mais nous n'en dirons pas plus pour le moment.

L'instruction END n'est d'ailleurs pas utile lorsque la fin logique du programme coïncide avec la dernière ligne du programme. L'ordinateur est en effet parfaitement capable de reconnaître la fin du programme.

Il vaut mieux cependant placer malgré tout cette instruction systématiquement à la fin de votre programme car c'est plus élégant et cela peut en outre vous éviter de légers problèmes si vous ajoutez plus tard des lignes de sous-programmes à votre programme.

Mais revenons pour l'instant à l'instruction PRINT, ou plutôt à une variante de cette instruction, l'instruction PRINT USING.

2.1.3.1 PRINT USING

Cette variante de l'instruction PRINT permet de réaliser des éditions "formatées" des valeurs ou textes que vous voulez sortir. Cette instruction est utilisée en liaison avec les caractères de commande suivants:

Pour les éditions numériques:

#	définit le nombre de chiffres à sortir
.	indique la position du point décimal
+	le signe + doit être sorti devant les nombres positifs
-	le signe - doit être sorti à la suite des nombres négatifs
**	remplir avec des étoiles à la place des espaces
$$	le premier caractère sorti doit être le signe $
**$	utilisation combinée de $ et *
,	une virgule de séparer par groupes de trois chiffres les chiffres placés à la gauche du point décimal
^^^^	les valeurs doivent être sorties en écriture scientifique

Pour les éditions de texte:

!	seul le premier caractère du texte ou de la variable alphanumérique doit être sorti
\ \	le nombre de caractères sortis correspondra au nombre d'espaces entre les traits plus 2
&	le texte sera sorti en entier

Illustrons l'utilisation de l'instruction PRINT USING par quelques exemples. Entrez d'abord dans votre ordinateur:

```
a = 12345.678          (appuyez sur la touche ENTER)
b = 34.3455 (ENTER)
c = -128    (ENTER)
d$= "CPC"   (ENTER)
```

Entrez maintenant les instructions PRINT USING suivantes en n'oubliant pas d'appuyer chaque fois sur la touche ENTER pour faire exécuter une instruction:

```
print using "#####.##";a
```

Vous obtenez en sortie sur l'écran la valeur de a arrondie à deux décimales:

```
12345.68
```

```
print using "#####.##";b
```

Vous obtenez en sortie sur l'écran la valeur de b arrondie à deux décimales:

```
   34.35
```

Notez que les points décimaux de ces deux valeurs sont parfaitement alignés, ce qui n'aurait pas été le cas si vous aviez utilisé uniquement l'instruction PRINT. Si le nombre à sortir sur l'écran dépasse le format indiqué dans l'instruction PRINT USING, un signe de pourcentage est affiché devant cette valeur pour indiquer que le format prescrit ne peut pas être respecté:

```
print using "####.##";a
```

Vous obtenez en sortie sur l'écran:

```
%12345.68
```

Si vous voulez souligner particulièrement le signe des résultats négatifs ou positifs, vous pouvez utiliser les combinaisons suivantes:

```
print using "+#####.##";a
```

sortie:

```
+12345.68
```

```
print using "#####.##+";a
```

sortie:

```
12345.68+
```

```
print using "#####.##-";c
```

sortie:

```
   128.00-
```

Le signe moins est ici placé à la suite de la valeur absolue du nombre alors qu'il est normalement placé devant elle.

Voici maintenant comment vous pouvez remplir les espaces non utilisés de votre format d'étoiles. Il s'agit d'un format que vous connaissez peut-être déjà pour l'avoir rencontré sur des relevés de compte:

```
print using "**#####.##";a
```

sortie:

```
**12345.68
```

```
print using "**#####.##";b
```

sortie:

```
*****34.35
```

Le format suivant vous permet d'indiquer des sommes en dollars:

```
print using "$$#####.##";a
```

sortie:

```
$12345.68
```

Vous pouvez bien sûr combiner le signe $ avec l'étoile:

```
print using "**$#####.##";b
```

sortie:

```
*****$34.35
```

Vous pouvez d'autre part séparer les groupes de trois chiffres par une virgule et ajouter également des symboles supplémentaires tels que F:

```
print using "**##,###.## F";a
```

sortie:

```
$12,345.68 F
```

Vous pouvez également faire éditer les nombres suivant l'écriture scientifique:

```
print using "##.##^^^^";a
```

sortie:

```
1.23E+04
```

Voyons maintenant les différents formats possibles pour la sortie de textes:

```
print using "!";d$
```

sortie:

```
C
```

Le point d'exclamation entraîne donc que seul le premier caractère de la variable alphanumérique (variable de texte) d$ sera sorti. Rappelez-vous que nous avons affecté le texte "CPC" à cette variable.
Le format suivant nous permet de définir le nombre de caractères d'une chaîne de caractères qui doivent être sortis:

```
print using "xx ";d$
```

sortie:

```
CPC
```

Les guillemets entouraient 3 caractères (en comptant les traits). C'est pourquoi seuls les trois premiers caractères de la variable d$ ont été sortis. Cet emploi de PRINT USING est proche de l'instruction LEFT$ dont nous parlerons plus loin.
Voici enfin comment faire imprimer l'intégralité d'un texte:

```
print using "&";d$
```

sortie:

```
 CPC
```

Notez que vous pouvez également arriver au même résultat si vous utilisez une instruction PRINT normale:

PRINT d$

Pour être complet, nous évoquerons encore l'instruction

WRITE

Il s'agit d'une autre variante de l'instruction PRINT. Elle se distingue par le fait qu'elle sort les guillemets avec le texte:

WRITE "CPC"

donnera à l'écran:

"CPC"

Si vous utilisez l'instruction WRITE avec des variables ou des constantes séparées par des virgules, la virgule n'aura pas de fonction de tabulation et sera sortie avec les variables et constantes:

WRITE 2,3,4

donnera à l'écran:

2,3,4

Nous avons maintenant décrit toutes les instructions qui constituaient notre petit programme ainsi que deux instructions "cousines" de ces instructions. Nous espérons que vous n'aurez plus de difficulté pour utiliser ces instructions.

Voyons maintenant comment l'instruction REM peut nous permettre de rendre nos programmes plus compréhensibles pour d'autres.

2.1.4 LES COMMENTAIRES AVEC L'INSTRUCTION REM

REM vous permet de placer des commentaires (REMarques) en n'importe quel endroit de votre programme. Tout ce qui suit une instruction REM est ignoré par l'ordinateur lors de l'exécution du programme, même s'il s'agit d'instructions BASIC. L'instruction REM est donc un très puissant outil de documentation des programmes.

Nous allons donc complèter maintenant notre programme d'exemple, de façon à le rendre parfaitement compréhensibles même pour ceux qui n'ont pas participé à son écriture:

```
10 REM CALCUL DE LA SURFACE D'UNE SPHERE
20 REM ENTREE DU RAYON EN CM
30 INPUT"QUEL RAYON EN CM";RAYON
40 REM CALCUL DE LA SURFACE
50 LET OFL=4*PI*RAYON^2
60 REM SORTIE DE LA SURFACE EN CM CARRES
70 PRINT"LA SURFACE DE LA SPHERE EST DE ";OFL;" CM 2"
80 END
```

Les lignes 10 et 20 indiquent le but du programme et comment les entrées doivent être faites. En ligne 30 les données sont entrées avec INPUT. Un message est envoyé à l'utilisateur du programme. La ligne 40 indique dans son commentaire le but de la ligne 50 qui est de calculer la surface à partir de la donnée entrée. La ligne 50 affecte à la variable S la surface calculée à partir de la formule mathématique placée à la droite du signe égale. La ligne 60 indique que la ligne suivante sort la surface en cm2. La ligne 70 sort la valeur de la variable S à la suite d'un texte explicatif. La ligne 80 termine le programme avec l'instruction END.

2.2 LES VARIABLES ET LEUR UTILISATION

DANS LES PROGRAMMES

Avant que nous ne vous donnions quelques problèmes à résoudre, il nous faut encore décrire les différents types de variables du BASIC de votre CPC.

Votre ordinateur connait en effet trois types de variables. Le premier type de variable est celui des variables entières (integer variable). Ce type de variable qui ne peut donc recevoir que des valeurs entières est désigné par le signe de pourcentage % placé à la suite du nom de la variable (par exemple A% ou C4%).

Si vous tentez d'affecter à une variable entière un nombre comportant des décimales, seule la partie entière de ce nombre sera prise en compte. Il faut cependant tenir compte d'une contrainte supplémentaire pour utiliser ce type de variable: non seulement les valeurs affectées doivent être entières mais elles doivent en outre être comprises entre -32768 et 32767.

Le second type de variable, variables réelles (real variable) permet de représenter tous les nombres. Les noms des variables de ce type n'ont pas à être désignées par un signe particulier. On peut toutefois utiliser à titre facultatif le signe d'exclamation ! à cet effet. Lorsque vous venez d'allumer votre ordinateur, toutes les variables numériques sont d'abord considérées comme des variables entières.

Le troisième type de variable est désigné par le signe dollar $. Il s'agit des variables alphanumériques ou chaînes de caractères. Ces variables peuvent recevoir n'importe quel texte mais elles ne peuvent contenir plus de 255 caractères.

Dans le choix des noms des variables il est important de tenir compte du fait que le BASIC du CPC autorise des noms de variables comportant jusqu'à 40 caractères. Vous pouvez donc parfaitement appeler une variable :
'SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS'.
Beaucoup d'ordinateurs n'acceptent pas de noms aussi longs. Cette caractéristique très importante de CPC vous permet de donner à vos variables des noms caractéristiques qui vous permettent de comprendre plus facilement leur rôle dans l'architecture générale de votre programme.
C'est ainsi que nous avons dans notre exemple appelé RAYON la variable contenant le rayon de la sphère. Vous ne pouvez pas utiliser les mots clés BASIC (instructions, variables du système, etc...) comme noms de variables mais vous pouvez les utiliser dans les noms de vos variables:

AND ne peut être une variable mais WAND peut parfaitement être le nom d'une variable. Vous trouverez une liste des mots clés réservés en annexe de cet ouvrage ou de votre manuel du CPC. Les chiffres peuvent également faire partie du nom des variables mais seulement à partir de la deuxième position: 1A ou 2emecompte ne peuvent pas être utilisés mais A1 ou compte9 sont parfaitement possibles comme noms de variables.

2.2.1 OPERATIONS DE CALCUL AVEC DES VARIABLES

Si vous voulez effectuer des calculs avec vos variables, il faut tout d'abord connaitre les rangs de priorité des différentes opérations de calcul:

Signe	Rang de priorité	Signification
^	premier rang	élévation à la puissance
*	second rang	multiplication
/	second rang	division
+	troisième rang	addition
-	troisième rang	soustraction

Les opérateurs logiques obéissent également à des règles de priorité: NOT a le premier rang de priorité, AND le second rang et OR le troisième rang après les opérateurs mathématiques qui passent de toute façon avant le opérateurs logiques.

Vous en savez maintenant suffisamment pour résoudre les exercices qui vous sont proposés dans les pages suivantes. Ces exercices comportent d'une part des questions sur les chapitres précédents et d'autre part des petits programmes à réaliser. Essayez d'abord de résoudre ces problèmes sans vous reporter aux chapitres précédents.

N'ayez surtout pas peur de faire des erreurs car la recherche des erreurs constitue de toute façon une partie importante de tout travail de programmation. La seule chose qui importe est que vous essayiez vraiment de faire ces exercices par vous-même puis que vous cherchiez à comprendre ensuite quelles erreurs vous avez faites éventuellement et pourquoi vous les avez faites. Nous vous invitons d'autre part à bien suivre les 5 étapes du travail de programmation que nous avons décrites. Avant d'entrer un nouveau programme, n'oubliez pas d'entrer l'instruction NEW qui supprime le programme en mémoire. Nous vous souhaitons beaucoup de succès dans votre travail sur ces exercices.

EXERCICES

1. Indiquez si les noms de variables suivants sont autorisés et justifiez votre point de vue:

a) X1	b) Date$	c) JOUR
d) OU%	e) IF	f) TIME$
g) 4Nombre%	h) 255	i) LUNDI

2. Ecrivez un programme qui entre quatre valeurs A, B, C et D et qui sort les valeurs A et B l'une derrière l'autre sur une seule ligne et les valeurs C et D dans la ligne suivante avec une tabulation.

3. Ecrivez un programme qui calcule la surface d'un triangle rectangle en mètres carrés et sort le résultat avec un message d'explication.

4. Ecrivez un programme qui fournisse le poids idéal d'un être humain (taille en cm moins 100 moins 10 pour cent). La taille doit être entrée en centimètres et le poids doit être affiché en kilogrammes.

5. Ecrivez un programme calculant le nombre de litres dans un aquarium après que le programme ait demandé qu'on lui indique les données en cm correspondant à la longueur, la hauteur et la largeur.

6. Modifiez le programme de l'exercice 2 de façon à ce que chaque valeur soit écrite dans une ligne distincte avec un texte d'accompagnement.

SOLUTIONS

1. a) valable
   b) valable
   c) valable
   d) valable
   e) IF non autorisé car c'est une instruction BASIC
   f) TIME$ non autorisé car c'est une variable du système
   g) 4nombre% non autorisé car le nom d'une variable ne doit pas commencer par un chiffre
   h) 255 non autorisé pour la même raison qu'en g)
   i) valable

2.

```
10 INPUT A,B,C,D
20 PRINT A;B
30 PRINT C,D
40 END
```

L'intérêt de l'exercice était de bien se rappeler la différence entre le point-virgule qui provoque une sortie directement l'une derrière l'autre de deux expressions et la virgule qui réalise une tabulation. Notez d'autre part, lorsque vous utilisez ce programme, que vous devez entrer autant de valeurs que l'ordinateur en attend, sinon le message REDO FROM START apparaît qui vous invite à recommencer votre entrée de données.

3.

```
10 REM ENTREE DE LA HAUTEUR ET
20 REM DE L'HYPOTHENUSE C EN METRES
30 INPUT"ENTREE H,C";hauteur,C
40 A=.5*hauteur*C
50 PRINT"LA SURFACE EST DE ";A;"M^2"
60 END
```

4.
```
10 INPUT"ENTREE TAILLE EN CM";CM
20 REM CALCUL POIDS IDEAL
30 IDEAL=CM-100
40 REM CALCUL 10 POUR CENT
50 POURCENT=IDEAL/100*10
60 IDEAL=IDEAL-POURCENT
70 PRINT"VOTRE POIDS IDEAL EST DE";IDEAL;"KG"
80 END
```

Cette exercice aurait également pu être résolu plus rapidement en réunissant en une seule ligne les lignes 30, 50 et 60:

```
30 IDEAL=(CM-100)-(CM-100)/100*10
```

Cette façon d'écrire aurait cependant pour gros inconvénient de ne pas permettre à une autre personne qui lirait le programme de comprendre rapidement et clairement de quel calcul il s'agit. Il est cependant vrai que cette façon d'écrire, si elle est moins claire, permet toutefois d'économiser de la place en mémoire. Il faut donc essayer d'arriver en permanence, en fonction de la capacité de mémoire disponible, à un compromis entre les exigences de la clarté (lignes de commentaires avec REM, aération du programme par des lignes sans instructions, une seule instruction par ligne) et la meilleure utilisation de la mémoire disponible.

Pour des programmes de moyenne importance, nous ne saurions donc trop vous recommander de ne pas hésiter à présenter les programmes de la façon la plus claire possible.

5.
```
10INPUT"HAUTEUR,LONGUEUR,PROFONDEURENCM",
haut,long,prof
20 REM CALCUL DU VOLUME
30 vol=haut*long*prof
```

```
40 REM CALCUL DES LITRES
50 vol=vol/1000
60 PRINT"CONTENU DE L'AQUARIUM";vol; "LITRES"
70 END
```

Ce programme entre d'abord les paramètres de hauteur, longueur et de profondeur de l'aquarium en centimètres. Vous voyez qu'il est très pratique de pouvoir donner des noms significatifs aux variables comme c'est le cas avec le CPC. La ligne 30 calcule ensuite le volume en centimètres cube. La ligne 50 divise le volume ainsi obtenu par 1000 pour obtenir la contenance de l'aquarium en litres.

6.

```
10 INPUT A,B,C,D
20 PRINT"A";A
30 PRINT"B";B
40 PRINT"C";C
50 PRINT"D";D
60 END
```

Nous espèrons que vous n'avez fait qu'une bouchée de ces exercices. Si cependant vous avez encore rencontré quelques difficultés avec un exercice ou un autre, relisez les passages des chapitres précédents qui se rapportent au problème posé dans ce ou ces exercices.

Le chapitre suivant abordera de nouvelles instructions et la façon de les utiliser dans vos programmes.

2.3 LES FONCTIONS NUMERIQUES

Nous allons aborder dans ce chapitre les fonctions mathématiques intégrées que possède votre ordinateur. Il nous faut pour cela faire un petit détour à travers le domaine des mathématiques. Ne vous inquiétez pas il ne s'agit pas pour nous de faire un cours de mathématiques, mais uniquement de vous permettre de comprendre à quoi servent ces fonctions mathématiques intégrées.

Dans de nombreux ouvrages sur le BASIC ainsi que dans le manuel de votre CPC, il est indiqué que les valeurs des fonctions trigonométriques SIN(X), COS(X) ou TAN(X) sont fournies conformément à la norme, en radian. Qu'est ce que cela signifie?

Il s'agit de l'indication de l'angle avec lequel le SINUS ou le COSINUS doit être calculé. Comme vous le savez certainement, un cercle est divisé 360 degrés. 1 degré est donc le 360ème d'un cercle. 90 degrés correspondent donc à un quart de cercle, 180 degrés à un demi-cercle, etc...

Pour calculer le radian, on ne divise pas le cercle en 360 degrés, mais on part de l'unité que représente le cercle, pour calculer la proportion du cercle représentée par un angle, d'après la formule:

U=2*PI*R

La formule est simplifiée en prenant un cercle unitaire (rayon=1):

U=2*PI*1 ou U=2*PI

Les angles sont donc ainsi mesurés non pas en degrés mais en "RAD" (radian). Dans notre exemple, le cercle possèderait 360 degrés ou 2*PI (6.2831...) RAD. 90 degrés correspondraient à 2*PI/4 ou PI/2 RAD.

L'intérêt de cette mesure des angles est qu'elle permet, à partir de la valeur obtenue pour un rayon de 1 de déterminer directement la longueur de l'arc de cercle. Ce type de calcul nécessite une certaine adaptation au départ car il est plus facile de se représenter un quart de cercle sous la forme de 90 degrés que de PI/2 RAD.

Cette brève incursion dans les mathématiques suffira pour l'instant. Voici maintenant quelques petits programmes d'exemple qui vous rendront ces concepts peut-être un peu plus clairs.

Entrez ce programme dans votre ordinateur:

```
10 CLS:DEG
20 INPUT"ENTREE EN DEGRES";DEGRES
30 REM CALCUL DU SINUS
40 SI=SIN(DEGRES)
50 PRINT"LE SINUS DE";DEGRES;"DEGRES ";
60 PRINT"EST =";SI
70 END
```

Lancez maintenant ce programme avec RUN et entrez pour l'angle la valeur 90. Vous devez obtenir 1 comme résultat. Le programme attend que vous entriez un angle exprimé en degrés et il calcule le sinus correspondant. Si vous voulez donc pouvoir entrer dans vos programmes des angles en degrés, vous devez utiliser la commutation en mesure des angles en degrés que réalise en ligne 10 l'instruction DEG. La ligne 10 utilise en outre l'instruction CLS (clear screen) qui efface le contenu de l'écran. Il est plus "propre" de faire commencer tout programme par cette instruction.

Pour calculer le cosinus, il suffirait de remplacer en ligne 40 SIN par COS. Vous pouvez d'ailleurs conserver le nom de la variable SI. Pour voir maintenant la différence avec les mesures en radian, entrez maintenant le programme suivant, non sans avoir au préalable fait exécuter l'instruction NEW:

```
10 CLS:RAD
20 INPUT"ENTREE EN RADIANS";RAD
30 REM CALCUL DU SINUS
40 SI=SIN(RAD)
50 PRINT"LE SINUS DE";RAD;"RADIANS ";
60 PRINT"EST =";SI
70 END
```

L'instruction RAD en ligne 10 est en principe inutile puisque le CPC commute à nouveau vers la mesure en radians si vous entrez NEW. Lancez le programme et entrez la valeur 1.57079633 (PI/2 = quart de cercle). Vous obtenez la valeur 0.

L'utilisation des autres fonctions est aussi simple. Il suffit de fournir à ces fonctions la valeur sur laquelle elles doivent effectuer un calcul. SQR(X) calcule la racine carrée de X, ATN(X) l'arc tangente de X, EXP(X) la Xième puissance de e=2.71828183 et LOG(X) le logarithme de X de base e. Ces deux dernières fonctions sont d'effet exactement inverse: si vous entrez:

```
PRINT EXP(1)
```

vous obtenez 2.71828183. Si vous entrez maintenant:

```
PRINT LOG(2.71828183)
```

vous obtenez à nouveau 1. Si vous voulez faire calculer le logarithme de base 10, il vous suffit de faire remplacer LOG(X) par LOG10(X).

Le logarithme de base e est aussi appelé logarithme naturel. Le programme d'exemple suivant calcule pour vous aussi bien le logarithme naturel que le logarithme de base 10:

```
10 CLS
20 INPUT"ENTREE DU NOMBRE";nombre
30 REM CALCUL DU LOGARITHME NATUREL
40 LN=LOG(nombre)
50 REM CALCUL DU LOGARITHME DE BASE 10
60 LO=LOG10(nombre)
70 PRINT"LE LOGARITHME NATUREL ";
80 PRINT"DE";nombre;" EST DE";LN
90 PRINT
100 PRINT"LE LOGARITHME DE BASE 10 ";
110 PRINT"DE";nombre;" EST DE";LO
120 END
```

Vous voyez que le maniement de ces fonctions dans des programmes (pour peu bien sûr qu'on sache à quoi elles correspondent en mathématiques) est très simple. La seule petite difficulté réside dans le traitement différent des degrés et des radians.

A RETENIR:

Les fonctions trigonométriques attendent des valeurs en radians. Pour effectuer un calcul en degrés, il faut commuter ces fonctions sur la mesure en degrés avec l'instruction DEG.

Les fonctions LOG et EXP se rapportent respectivement à l'exposant et à la base e.

La fonction SGN(X) fournit le signe de X. Le résultat est 1 pour un nombre positif, 0 pour la valeur 0 et -1 pour un nombre négatif.

La fonction INT(X) fournit la partie arrondie de X. Avec une petite routine utilisant cette instruction, il est donc facile d'arrondir des nombres au nombre de décimales que vous voulez. C'est ce que réalise le petit programme suivant:

```
10 CLS
20 INPUT"COMBIEN DE CHIFFRES APRES LA VIRGULE";X%
30 INPUT"QUEL NOMBRE";nombre
40 REM ARRONDIR
50 nombre=INT(nombre*10^X%+.5)/10^X%
60 REM SORTIE DU NOMBRE ARRONDI
70 PRINT nombre
80 END
```

La ligne 20 vous demande à combien de décimales devra être arrondi le nombre. Le nombre de décimales est affecté à la variable X%. Nous utilisons une variable entière puisqu'il ne peut pas y avoir de nombre non entier de décimales. La ligne 30 vous demande alors d'entrer un nombre quelconque. Entrez ici n'importe quel nombre décimal dont le nombre de décimales soit supérieur au nombre de décimales à laquelle vous voulez l'arrondir.

La ligne 50 arrondit ce nombre. Ce nombre est d'abord multiplié par 10 puissance X%, ce qui place les décimales que vous voulez conserver à la gauche du point décimal. .5 (1/2) est alors ajouté au nombre à arrondir car la fonction INT se contente de supprimer les décimales en trop, c'est-à-dire d'arrondir systématiquement à l'unité inférieure. En ajoutant .5, vous êtes sûr que votre nombre sera arrondi à l'unité la plus proche. Le nombre est enfin divisé par 10 puissance X% ce qui donne comme résultat final votre nombre arrondi au nombre de décimales que vous avez indiqué.

Lancez le programme avec RUN (ENTER). Entrez quelques valeurs pour voir à quels résultats vous arrivez. Essayez surtout de bien comprendre la logique de la ligne 50 qui réalise le calcul de la valeur arrondie. Vous pourrez ainsi mieux comprendre tout ce que doit faire votre CPC lorsque vous utilisez l'instruction ROUND.

'ROUND' vous permet en effet d'arrondir très facilement n'importent quels nombres ou variables. La syntaxe, le format de cette instruction est le suivant:

ROUND(X,Y)

X étant le nombre à arrondir et Y le nombre de décimales auquel il doit être arrondi

PRINT ROUND(3.12354,3)

donnera:

3.124

Le petit programme d'exemple que nous vous avons fourni plus haut n'a donc pas d'utilité pratique sur un CPC. Mais il vous montre comment on peut pallier l'absence de certaines instructions en développant soi-même des routines équivalant à ces instructions. Vous pourrez d'ailleurs avoir besoin de cette routine si vous travaillez sur d'autres ordinateurs, car tous les ordinateurs sont loin de posséder un BASIC aussi puissant que le CPC.

Evoquons encore brièvement une autre instruction pour arrondir des nombres:

CINT

Cette instruction arrondit les nombres ou les variables comme INT mais elle convertit en outre la valeur arrondie en format entier. Les nombres ainsi convertis doivent donc être entre -32768 et 32767.

L'instruction

CREAL

est symétrique de l'instruction CINT puisqu'elle convertit une variable entière en format réel.

La fonction MOD fournit le reste d'une division:

PRINT 32 MOD 7

donnera le reste de 32 divisé par 7, soit 4.

2.3.1 FONCTIONS AVEC DEF FN

L'instruction DEF FN vous fournit un moyen pratique d'économiser de la place en mémoire. Cette instruction permet d'affecter des fonctions mathématiques complexes à l'expression FN. Cette expression peut en effet être appelée ensuite pour calculer la valeur correspondant au paramètre qui lui est fourni, d'après la fonction qui a été définie avec DEF FN.

Voici un exemple:

```
10 REM DEFINITION DE LA FONCTION
20 DEF FN(X)=X^2 + 2*X + 4
30 REM ENTREE DU PARAMETRE
40 INPUT"QUELLE VALEUR";X
50 REM SORTIE
60 PRINT FN F(X)
70 END
```

La ligne 20 affecte la fonction mathématique X^2+2X+4 à l'expression FN F(X). X représente le paramètre en fonction duquel le résultat FN F(X) sera calculé. Vous n'avez donc à écrire qu'une fois toute fonction aussi complexe soit elle dont vous aurez à vous servir souvent dans votre programme. Il vous suffit ensuite d'utiliser le nom de la fonction.

2.3.2 NOMBRES ALEATOIRES

Le BASIC du CPC possède un générateur de hasard intégré que vous pouvez appeler avec l'instruction RND(X). Cette fonction peut vous permettre d'effectuer une simulation où le hasard joue un rôle ou bien sûr aussi de provoquer des évènements imprévisibles dans un programme de jeu. L'utilisation de cette fonction est très simple: A=RND(1) affecte à A une valeur aléatoire entre 0.0 et 1.0. Si vous utilisez comme valeur de X (entre parenthèses) des valeurs négatives, c'est toujours la même suite de nombre qui sera sortie. Le programme suivant simule un jet de dé. Chaque fois que vous lancez le programme, vous obtenez un nombre entre 1 et 6:

```
10 REM PRODUIRE UN NOMBRE ALEATOIRE
20 A=INT(6*RND(1))+1
30 PRINT A
40 END
```

Si vous lancez ce programme plusieurs fois, vous ne pourrez constater aucune régularité dans la sortie des différents nombres.

La ligne 20 utilise la fonction INT pour qu'il n'y ait pas de nombres avec décimales. Le nombre aléatoire est multiplié par 6 ce qui permet d'obtenir une valeur entre 0.0 * 6 = 0 et 1.0 * 6 = 6. Il s'agit là de bornes qui ne sont jamais atteintes. On a encore ajouté 1 pour avoir non pas des nombres compris entre 0 et 5 inclus mais entre 1 et 6 inclus.

Si vous vouliez maintenant obtenir des nombres aléatoires entre 100 et 150, il faudrait modifier ainsi la ligne 20:

```
20=INT(150*RND(1))+100
```

Voici la formule générale pour obtenir un nombre compris entre "plancher" et "plafond":

```
A=INT((plafond-plancher)*RND(1))+plancher
```

Si vous voulez que le générateur de hasard soit initialisé avec une nouvelle valeur de départ, vous pouvez utiliser l'instruction RANDOMIZE:

```
RANDOMIZE 45
```

affecte au générateur de hasard la nouvelle valeur de départ 45. Vous pouvez maintenant affecter à la variable A une nouvelle valeur aléatoire avec l'instruction:

```
A=RND(4)
```

2.3.3 AUTRES INSTRUCTIONS POUR MANIER

LES VARIABLES

Le riche BASIC du CPC possède toute une série d'instructions permettant de définir, de modifier ou de supprimer des variables ou des formats de variables. Voici une brève description de ces instructions:

Souvenez-vous un instant du chapitre sur les systèmes numériques. Le CPC offre deux instructions permettant de convertir les nombres décimaux en nombres binaires ou en nombres hexadécimaux.

L'instruction

BIN$(X,Y)

convertit un nombre décimal en un nombre binaire. X est le nombre à convertir et Y indique le format dans lequel ce nombre doit être converti:

PRINT BIN$(24,8)

donnera:

00011000

L'instruction

HEX$(X)

convertit un nombre décimal en un nombre hexadécimal. X est le nombre à convertir:

PRINT HEX$(60)

donnera:

3C

Comme nous l'avons déjà indiqué, le BASIC du CPC connait trois différents types de variables (entières, réelles et alphanumériques). Si vous voulez définir à quel type de variables doivent appartenir certaines variables d'un programme, vous pouvez utiliser les instructions suivantes:

DEFINT

DEFSTR

DEFREAL

Si vous entrez par exemple:

DEFINT A-B

toutes les variables commençant par A ou B seront maintenant définies comme étant des variables entières. Vous n'aurez donc plus besoin de placer à la suite du nom de ces variables le signe pourcentage (%). Les autres instructions fonctionnent de façon analogue.

Faites toutefois attention en utilisant ces instructions qui réclament de votre part une bonne connaissance des noms des variables utilisées dans votre programme. En effet si vous oubliez par exemple que vous avez défini les variables A à B comme variables entières et que vous essayez dans le cours du programme d'affecter une chaîne de caractères à la variable A, votre programme s'interrompra pour afficher le message d'erreur:

Type mismatch in (numéro de la ligne)

Si vous ne voulez sortir que la partie d'une valeur située à la gauche du point décimal, vous pouvez utiliser l'instruction:

FIX

La différence entre FIX et INT est que FIX supprime systématiquement les décimales alors que INT arrondit les nombres négatifs au premier nombre entier négatif plus petit:

PRINT INT(-3.55)

donnera:

-4

alors que

PRINT FIX(-3.55)

donnera:

-3

Deux instructions puissantes vous permettent de déterminer le plus grand nombre ou le plus petit nombre figurant dans une liste de nombres. L'instruction

MAX

fournit le plus grand nombre d'une liste de nombres. Cette liste peut également comprendre des variables:

PRINT MAX(23,4,65,67,123,344,33)

donnera:

344

L'instruction

MIN

fournit par contre le plus petit nombre d'une liste de nombres. Cette liste peut également comprendre des variables:

Si vous entrez:

Z=-56

PRINT MIN(3,4,1,Z,-3,-25)

donnera:

-56

Nous avons maintenant fait le tour des principales instructions de travail avec les variables.

2.3.4 ASC(X$) ET CHR$(X)

Le CPC vous permet de sortir sur l'écran, avec l'instruction PRINT des nombres, des lettres et certains caractères graphiques. Il existe cependant également des caractères de commande qui vous permettent par exemple de définir la couleur des caractères suivants ou de faire représenter des textes en inversion vidéo. Ces caractères de commande qui sont le plus souvent définis par des caractères graphiques quelconques ont l'intérêt de vous permettre d'obtenir certaines fonctions, lorsque vous écrivez des programmes, simplement en appuyant sur une touche. Ces caractères de commande ont cependant également un gros inconvénient: ils sont souvent difficiles à distinguer entre eux, de sorte qu'il n'est pas toujours simple de les reconnaître à la lecture du listing d'un programme.

D'autre part la forme de ces caractères peut également varier en fonction de l'imprimante utilisée. Ceci est certainement peu gênant pour celui qui a écrit le programme et qui devrait en principe se souvenir des caractères de commande qu'il a utilisés mais pour une personne qui veut simplement "taper" un programme pour l'entrer dans l'ordinateur d'après un listing tout fait, il risque d'être difficile de ne pas commettre d'erreurs dans l'interprètation des différents symboles figurant sur le listing.

C'est pourquoi l'utilisation des codes CHR$ peut vous simplifier considérablement le travail. Par exemple, pour déclencher un bip, vous pouvez entrer:

PRINT" "

Pour obtenir ce signe, il faut appuyer simultanément sur les touches CTRL et G.

Mais vous pouvez aussi, pour plus de clarté, utiliser l'instruction:

PRINT CHR$(7)

qui sera facilement identifiable par tout lecteur éventuel du listing et que vous-même n'aurez aucun mal à reconnaître si vous réexaminez votre listing après plusieurs semaines d'interruption.

Il en va de même pour les caractères graphiques correspondant aux caractères de commande du curseur ou de la touche ENTER. Ces caractères se ressemblent en effet tellement qu'il est toujours difficile de les distinguer. Il est donc préfèrable d'utiliser d'une manière générale les codes CHR$ à la place des caractères de commande.

Vous trouverez en annexe de cet ouvrage ou du manuel de votre CPC les codes CHR$ de tous les caractères. Ces codes, qui fournis à la fonction CHR$, donnent le caractère graphique ou la lettre correspondants, sont les mêmes codes que les codes ASCII dont nous avons déjà parlé. Vous pouvez à tout moment indiquer à votre ordinateur de vous indiquer le code ASCII d'un caractère quelconque:

PRINT ASC("A")

donnera: 65 qui est le code ASCII de la lettre A. La fonction CHR$ permet donc d'obtenir l'effet inverse:

PRINT CHR$(65)

donnera un A sur l'écran.

En ce qui concerne les caractères de commande, nous essaierons de ne plus utiliser dans les prochains listings d'exemples que nous vous fournirons que les codes CHR$. Ces codes présentent encore un avantage que nous n'avons pas évoqué: ils facilitent en général la transposition de programmes d'un ordinateur à l'autre.

Si les caractères graphiques correspondant aux divers caractères de commande sont en effet souvent différents d'un ordinateur à l'autre, il existe par contre certaines normes pour les codes ASCII de ces caractères. C'est ainsi que PRINT CHR$(7) produira sur beaucoup d'ordinateurs le même effet (un bip) que sur le CPC. Vous pouvez donc retranscrire sans la modifier cette instruction si vous la rencontrez lors d'un travail de transposition d'un programme d'un ordinateur sur un autre.

Voici maintenant quelques exercices pour vous permettre d'utiliser votre nouvelle science. Ces exercices vous permettront en effet d'employer les instructions qui viennent d'être expliquées. Nous vous recommandons à nouveau de bien respecter les cinq étapes du travail de programmation et nous vous souhaitons beaucoup de réussite.

EXERCICES

1. Ecrivez un programme qui simule le jet de dès avec deux dés. Le résultat des deux dés doit être affiché sur une seule ligne avec une tabulation. N'oubliez pas de faire vider l'écran en début de programme.

2. Ecrivez un programme qui calcule la surface d'un triangle quelconque d'après la formule:
F=SQR(S(S-A)(S-B)(S-C))ouS=1/2(A+B+C).Réfléchissezaufait que vous ne pouvez pas entrer cette formule dans votre programme sous cette forme. Le programme doit demander l'entrée des valeurs A, B et C. Le résultat doit être présenté par un message d'explication.

3. Ecrivez un programme qui vous demande d'entrer un caractère et qui sorte ensuite ce même caractère suivi de son code ASCII.

4. Ecrivez un programme qui calcule une hauteur d'après le temps de chute d'un corps. Le programme doit demander l'entrée du temps de chute mesuré. La résistance de l'air ne sera pas prise en compte. La formule est S=1/2GT^2. La constante G vaut 9.81. Le résultat sera sorti en mètres.

5. Ecrivez un programme qui calcule la consommation d'essence aux 100 kilomètres d'après la formule:

   Consommation aux 100 kilomètres = consommation totale / nombre de kilomètres parcourus * 100

SOLUTIONS

1.

```
10 REM VIDER L'ECRAN
20 CLS
30 REM GENERER DES NOMBRES ALEATOIRES
40 DE1=INT(6*RND(1))+1
50 DE2=INT(6*RND(1))+1
60 REM SORTIE DU RESULTAT
70 PRINT"DE 1:";DE1,"DE 2:";DE2
80 END
```

Voici à quoi votre programme devrait ressembler. Les solutions qui vous sont proposées ici ne sont toutefois que des propositions car tous les chemins mènent à Rome et la seule chose qui compte en informatique, c'est le résultat. Si vous lancez plusieurs fois ce programme avec RUN, vous obtiendrez chaque fois des valeurs différentes.

La ligne 20 vide l'écran. Les lignes 40 et 50 affectent aux variables DE1 et DE2 deux nombres aléatoires. Si vous avez eu du mal à définir les bornes inférieure et supérieure des nombres aléatoires, relisez le chapitre sur les nombres aléatoires.

2.

```
10 REM ENTREE DES COTES DU TRIANGLE
20 INPUT"ENTREZ A,B,C EN CM";A,B,C
30 REM CALCUL DE S
40 S=.5*(A+B+C)
50 REM CALCUL DE LA SURFACE
60 F=SQR(S*(S-A)*(S-B)*(S-C))
70 REM SORTIE DE LA SURFACE
80 PRINT"LA SURFACE DU TRIANGLE ";
90 PRINT"EST DE";F;" CM^2"
100 END
```

Faites attention dans ce programme à calculer d'abord la valeur S puisque cette variable est utilisée pour le calcul de la surface. La traduction en BASIC de la formule ne devrait pas vous avoir présenté de difficulté. Notez simplement que vous ne pouvez pas écrire les formules mathématiques sous leur forme classique dans vos programmes. Si vous le faites, vous risquez de provoquer une interruption de votre programme et un message SYNTAX ERROR, chaque fois que l'écriture classique se distingue de l'écriture BASIC.

3.
```
10 INPUT"APPUYEZ SUR UN TOUCHE";A$
20 A=ASC(A$)
30 PRINT"LE CODE ASCII DE ";A$;" EST";A
40 END
```

Si vous avez essayé ce programme et tenté d'obtenir ainsi le code ASCII de la touche ENTER ou de la virgule, vous aurez vu apparaître un message IMPROPER ARGUMENT IN 20. L'instruction INPUT présente en effet l'inconvénient de n'utiliser la virgule que pour séparer entre elles des données entrées. Si vous appuyez par ailleurs sur la touche ENTER, l'ordinateur considère que vous êtes en train de valider votre entrée et que vous n'avez donc rien entré. C'est la chaîne vide qui est donc affectée à la variable alphanumérique. Comme l'ordinateur ne peut évidemment pas calculer le code ASCII d'une chaîne vide, il vous envoie un message d'erreur. Nous expliquerons plus loin, à propos de l'instruction INKEY, comment il est possible de pallier cette difficulté.

4.
```
10 G=9.81
20 INPUT"COMBIEN DE SECONDES";T
30 S=.5*G*T^2
40 PRINT"LE CORPS EST TOMBE D'UNE";
50 PRINT" HAUTEUR DE";S;" METRES"
60 FIN
```

La ligne 10 qui affecte à la variable G la valeur 9.81 réalise ce qu'on appelle l'INITIALISATION D'UNE VARIABLE. Ce procédé qui consiste à affecter en début de programme différentes valeurs à des variables permet dans la suite du programme de ne plus avoir à utiliser que les variables au lieu de ces valeurs qui peuvent parfois comporter beaucoup de chiffres. Dans de longs programmes, c'est là un moyen de gagner de la place en mémoire qui ne doit pas être négligé.

5.
```
10 INPUT"CONSOMMATION EN LITRES";lit
20 INPUT"NOMBRE DE KILOMETRES PARCOURUS";km
30 consommation=lit/km*100
40 PRINT"CONSOMMATION AUX 100 KM";
consommation;" LITRES"
50 END
```

Ce programme ne nécessite pas de commentaire particulier.

Si vous avez résolu ces différents problèmes sans difficulté, vous pouvez maintenant passer aux chapitres suivants. Dans le cas contraire, nous vous invitons à relire les passages correspondant aux problèmes que vous avez eu du mal à résoudre.

2.4 TAB, SPC ET ZONE

Ces trois instructions permettent d'afficher des données ou des caractères dans des emplacements déterminés de l'écran. L'instruction TAB et son paramètre entre parenthèses positionnent le caractère à sortir relativement au début de la ligne de l'écran où se trouve actuellement le curseur:

```
PRINT TAB(15) "TEST"
```

placera le mot TEST à partir du 15ème caractère de la ligne de l'écran. Si vous placez le curseur en début de ligne, vous pouvez appuyer 14 fois sur la touche curseur droite, et vous verrez que le curseur se trouve alors en-dessous de la première lettre du mot TEST. Utilisez maintenant l'instruction SPC de la même façon, à la place de l'instruction TAB. Vous obtenez presque le même résultat. Prenons donc un exemple qui montre mieux la différence entre ces deux instructions. Videz l'écran avec CLS. Puis entrez:

```
PRINT TAB(5) "TEST 1" TAB(20)"TEST 2"
```

Vous voyez que le mot TEST 1 se trouve à partir de la cinquième position de la ligne, et le mot TEST 2 à partir de la vingtième. Remplacez maintenant TAB par SPC:

```
PRINT SPC(5) "TEST 1" SPC(20)"TEST 2"
```

Vous voyez tout de suite la différence: le mot TEST 2 n'a pas été affiché à partir de la vingtième position de la ligne mais à partir de la vingtième position après le dernier caractère de TEST 1. TAB se réfère donc à une position absolue sur la ligne de l'écran, et SPC à une position relative par rapport au dernier caractère sorti.

Si vous utilisez ces instructions pour une sortie sur imprimante, notez que l'instruction TAB n'a pas d'intérêt dans ce cas car, en liaison avec l'instruction PRINT # elle sera ignorée ou interprétée comme l'instruction SPC. TAB ne doit donc être utilisé qu'avec des PRINT "normaux".

A RETENIR

TAB(X) compte le nombre d'emplacements à partir du premier emplacement de la ligne de l'écran actuelle

SPC(X) compte ajoute en fait X espaces avant que ne reprenne la sortie

Nous avons déjà indiqué que l'écran du CPC est divisé en zones de 13 caractères. Pour modifier cette valeur, vous pouvez utiliser l'instruction ZONE:

ZONE 10

fixera le tabulateur à 10 caractères. Pour voir la différence avec la situation standard (qui est rétablie chaque fois que vous allumez votre ordinateur), vous pouvez essayer l'exemple qui était donné dans le chapitre sur l'instruction PRINT.

2.5 LES CHAINES DE CARACTERES

Une chaîne de caractères peut comporter 255 caractères sur le CPC. Les variables contenant des chaînes de caractères, les variables alphanumériques sont marquées par le signe dollar $. Pour affecter une chaîne de caractères à une variable alphanumérique, vous pouvez procéder exactement comme pour les variables numériques, avec cette

seule différence que le texte de la chaîne de caractères doit être absolument placé entre guillemets:

A$="CPC"

Si vous essayez d'affecter une valeur numérique à une variable alphanumérique, vous obtenez le message d'erreur:

Type mismatch

Vous obtiendrez d'ailleurs le même message d'erreur si vous essayez d'affecter une chaîne de caractères à une variable numérique:

A="TEST"   (ERREUR)!

Le seul opérateur de calcul qui puisse être utilisé en liaison avec les chaînes de caractères, est le signe plus +. Ce signe permet de "concaténer" deux chaînes de caractères. Si vous définissez A$="LECTEUR DE " et B$="DISQUETTE",
A$+B$ donnera "LECTEUR DE DISQUETTE":

```
10 A$="LECTEUR DE ":B$="DISQUETTE"
20 LD$=A$+B$
30 PRINT LD$
40 END
```

La ligne 10 initialise les variables A$ et B$. La ligne 20 affecte le résultat de la concaténation de A$ et B$ la variable LD$. La ligne 30 sort la chaîne ainsi obtenue.

Vous ne pouvez pas seulement concaténer différentes chaînes de caractères, vous pouvez également les comparer entre elles d'après leur place dans l'ordre alphabétique ou leur nombre de caractères. Nous reviendrons sur cette possibilité lorsque nous traiterons des instructions de comparaison (voir IF ... THEN).

Notez simplement pour le moment que vous ne pouvez comparer entre elles des variables de différents types: vous ne pouvez comparer des valeurs numériques à des chaînes de caractères.

Le BASIC du CPC vous permet également de manipuler les chaînes de caractères. C'est aux instructions qui permettent ces manipulations que nous allons maintenant nous intéresser.

L'instruction LEFT$ permet de constituer une chaîne de caractères à partir d'une partie d'une chaîne déterminée. Pour plus de clarté, veuillez taper maintenant le programme suivant:

```
10 A$="COMPUTER"
20 B$=LEFT$(A$,1)
30 C$=LEFT$(A$,2)
40 D$=LEFT$(A$,3)
50 E$=LEFT$(A$,4)
60 F$=LEFT$(A$,5)
70 G$=LEFT$(A$,6)
80 H$=LEFT$(A$,7)
90 I$=LEFT$(A$,8)
100 PRINT A$:PRINT B$:PRINT C$:PRINT D$
110 PRINT E$:PRINT F$:PRINT G$:PRINT H$:PRINT I$
120 END
```

Lancez maintenant ce programme avec RUN. Vous obtenez:

```
C
CO
COM
COMP
COMPU
COMPUT
COMPUTE
COMPUTER
```

Cet exemple montre bien le fonctionnement de l'instruction LEFT$. La ligne 10 affecte à la variable A$ la chaîne de caractères "COMPUTER". La ligne 20 constitue une sous-chaîne de A$ en prenant le premier caractère de A$ en partant de la gauche. Cette sous-chaîne est affectée à la variable B$. La ligne 30 constitue également une sous-chaîne de A$ en prenant cette fois les deux premiers caractères de A$ en partant de la gauche. Cette sous-chaîne est affectée à la variable C$. Les lignes 40 à 90 fonctionnent de la même façon.

L'instruction LEFT$(A$,X) crée donc une chaîne avec les X premiers caractères de A$ en partant de la gauche. Les lignes 100 à 110 permettent de sortir les nouvelles chaînes constituées. Nous avons cédé ici à la facilité qui consiste à écrire plusieurs instructions sur une même ligne parce que ces deux lignes restent malgré tout parfaitement compréhensibles.

Comme vous le voyez cette instruction peut être très utile pour obtenir des effets dans des jeux mais elle peut aussi déboucher sur des applications très sérieuses, notamment dans le domaine du traitement de données.

Etudions maintenant l'instruction RIGHT$ dont les effets sont très proches de ceux de l'instruction LEFT$.
En effet cette instruction RIGHT$ permet également de constituer une sous-chaîne d'une chaîne donnée, mais en prenant les caractères en partant de la droite de la chaîne de référence, au lieu de partir de la gauche. Si vous remplacez maintenant toutes les instructions LEFT$ de notre exemple précédent par des instructions RIGHT$, vous obtiendrez après avoir lancé le programme:

```
R
ER
TER
UTER
```

PUTER
MPUTER
OMPUTER
COMPUTER

Vous pouvez également modifier le programme à nouveau en faisant varier les paramètres de l'instruction RIGHT$ de 8 à 1 au lieu qu'ils varient de 1 à 8; vous obtenez alors l'image contraire. Ces exemples ont pour simple but de vous montrer le fonctionnement de cette instruction, de façon à ce que vous puissiez ensuite donner libre cours à votre fantaisie dans l'emploi de cette instruction.

Une autre instruction très intéressante de manipulation des chaînes de caractères est l'instruction MID$. Voici tout de suite un exemple d'application de cette instruction qui nous permettra de comprendre plus aisément sa fonction:

```
10 A$="SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS"
20 B$=MID$(A$,1,7)
30 C$=MID$(A$,8,9)
40 D$=MID$(A$,20,12)
50 E$=MID$(A$,32,8)
60 PRINT A$
70 PRINT B$
80 PRINT C$
90 PRINT D$
100 PRINT E$
110 END
```

Lancez maintenant le programme et examinez le résultat. Vous voyez donc que l'instruction MID$ vous permet de constituer des sous-chaînes d'une chaîne de caractères, en prenant le nombre que vous voulez de caractères de cette chaîne à partir du caractère de cette chaîne que vous fixez:

MID$(A$,X,Y)

A$ est le nom de la chaîne de départ. X est la position dans la chaîne de départ du premier caractère à partir duquel doit être constituée la sous-chaîne et Y est le nombre de caractères qui doivent constituer cette sous-chaîne. C'est ainsi que l'instruction MID$ en ligne 20 crée à partir de "societenationaledescheminsdeferfrancais" une sous-chaîne de 7 caractères commençant par le premier caractère de cette expression. Dans ce cas, si le premier paramètre de cette instruction est égal à 1 vous voyez qu'on aboutit au même résultat qu'avec l'instruction LEFT$(A$,1,7). Nous avons ensuite fait un usage plus normal de cette instruction à partir de la ligne 30 qui affecte par exemple à C$ la sous-chaîne de A$ composée de 9 caractères de A$ à partir du 8ème caractère de A$.

Notez que cette instruction est très puissante puisque vous pouvez utiliser comme paramètres aussi bien des variables ou des expressions arithmétiques que des constantes. Vous pouvez également utiliser cette instruction pour modifier le contenu d'une chaîne de caractères:

MID$(A$,7,1)="a"

transformera "societe" en "societa".

Avant de passer à une autre instruction, un petit exercice: comptez combien de caractères comporte notre expression A$("societenationaledescheminsdeferfrancais"). Si vous avez bien compté, vous devez en trouver 39.

Le seul but de ce petit exercice était de vous faire entrevoir l'intérêt de l'instruction que nous allons maintenant étudier, l'instruction LEN(X$) qui fournit la longueur d'une chaîne de caractères. Si vous entrez maintenant:

PRINT LEN(A$)

vous voyez apparaître 39. Notez que tous les caractères d'une chaîne de caractères sont pris en compte par LEN(X$), y compris les espaces.

L'instruction VAL(X$) est une autre instruction de maniement des chaînes de caractères qui vous permet de transformer toute chaîne de caractères en une expression numérique. Cette instruction vous permet surtout de transformer les nombres qui se trouvent pour une raison ou une autre sous forme d'une chaîne de caractères en une valeur numérique avec laquelle vous allez pouvoir effectuer des calculs.

Si le premier caractère de la chaîne de caractère dont vous voulez connaître la valeur numérique avec cette instruction est un caractère qui ne peut être interprété comme un chiffre, la valeur obtenue sera 0. Si un caractère qui ne peut pas être interprété comme un chiffre se trouve à l'intérieur de la chaîne de caractères considérée, la valeur obtenue sera le nombre constitué par les chiffres placés avant ce caractère non numérique. Voici quelques exemples qui vous permettront de mieux comprendre le fonctionnement concret de cette instruction:

EXEMPLES

a)
```
10 A$="343.45"
20 A=VAL(A$)
30 PRINT A
```

Résultat:
343.45

b)
```
10 B$="D34.87F"
20 B=VAL(B$)
30 PRINT B
```

Résultat:
0

c)
```
10 C$="234FFC54"
20 C=VAL(C$)
30 PRINT C
```

Résultat:
234

d)
```
10 D$="33,21"
20 D=VAL(D$)
30 PRINT D
```

Résultat:
33

Entrez ces petits exemples dans votre ordinateur et essayez-les. L'exemple a) vous montre un cas où la chaîne de caractère peut être entièrement transformée en une expression numérique. La chaîne B$ commence par contre par un caractère non numérique et sa valeur est donc nulle. C$ montre une chaîne mixte dont seule la partie numérique placée avant un caractère non numérique sera prise en compte pour le calcul de la valeur de la chaîne. Le dernier exemple a simplement pour but de vous rappeler qu'en BASIC, contrairement aux mathématiques européennes, la virgule ne peut être interprétée comme un caractère numérique. C'est le point qui en tient lieu, comme dans notre premier exemple.

L'instruction VAL(X$) est exactement symétrique de l'instruction STR$(X) qui convertit en effet toute valeur numérique en une chaîne de caractères pouvant donc être traitée comme telle avec les nombreuses instructions de maniement des chaînes de caractères que nous avons vues plus haut. Notez que le premier caractère de la chaîne ainsi créée ne sera pas le premier chiffre de la valeur numérique à convertir mais le signe de cette valeur numérique, donc un espace pour un nombre positif et le signe moins pour un nombre négatif.

EXEMPLES

a)
```
10 A=1234
20 A$=STR$(A)
30 PRINT A$
```

Résultat:
```
 1234
```

b)
```
10 B=-1234
20 B$=STR$(B)
30 PRINT B$
```

Résultat:
```
-1234
```

Les chaînes créées par nos deux exemples comportent donc toutes deux 5 caractères. Comme c'est le cas de toutes les intructions BASIC, le paramètre de cette instruction peut être une variable ou une expression arithmétique.

Voici maintenant une dernière instruction importante pour manipuler les chaînes de caractères, l'instruction:

INSTR

qui vous permet de rechercher si une suite de caractères quelconque se trouve dans une chaîne de caractères:

INSTR(X,A$,B$)

A$ est la chaîne de caractères dans laquelle doit être recherchée la suite de caractères B$. X est la position du caractère de A$ à partir duquel doit commencer la recherche. Le résultat obtenu est la position de la suite de caractère B$ dans la chaîne A$. Ce résultat est nul si la chaîne recherchée n'a pas été trouvée.

Voici un petit programme d'exemple:

```
10 A$="societenationaledescheminsdeferfrancais"
20 B$="nationale"
30 Z=INSTR(1,A$,B$)
40 PRINT Z
50 END
```

Lancez ce programme avec RUN et vous obtiendrez 8 comme valeur de Z. Notez que cette instruction recherche exactement la suite de caractères que vous avez indiquée. Ceci signifie donc que dans notre petit programme, l'ordinateur n'aurait pas trouvé l'expression "Nationale" parce que "Nationale" ne figure pas avec une majuscule dans A$.

A RETENIR:

Si vous utilisez INSTR(X,A$,B$), X doit toujours être supérieur à 0. Sinon vous obtenez le message d'erreur:
'IMPROPER ARGUMENT IN (numéro de ligne).

La suite de caractères que vous faites rechercher doit correspondre exactement à celle qui figure dans la chaîne que vous faites examiner. Faites attention aux majuscules et aux minuscules.

Voici maintenant deux autres instructions qui facilitent quelque peu le maniement des chaînes de caractères:

UPPER$

et

LOWER$

UPPER$ transforme tous les caractères d'une chaîne de caractères en majuscules et LOWER$ transforme tous les caractères d'une chaîne de caractères en minuscules. Voici un exemple simple:

```
A$=CPC"          (ENTER)
PRINT LOWER$(A$)  (ENTER)
```

donnera:

CPC

Voici maintenant une instruction qui peut notamment vous permettre d'améliorer la présentation de vos programmes dont nous parlerons plus loin. Cette instruction:

STRING$

crée une chaîne de caractères composée du même caractère:

```
A$=STRING$(40,"*")  (ENTER)
PRINT A$            (ENTER)
```

Vous obtenez une ligne de 40 étoiles. Ceci peut vous permettre également de réaliser aisément des masques écran, mais c'est un point sur lequel nous reviendrons plus loin.

L'instruction:

SPACE$

équivaut à l'instruction STRING$(X," ").

Elle crée donc une chaîne composée d'espaces:

A$=SPACE$(20)

créera une chaîne de 20 espaces. Cette instruction peut être utilisée pour positionner un texte sorti sur l'écran:

PRINT SPACE$(10);"CPC"

donnera:

```
          CPC
```

Voici maintenant quelques exercices pour vous permettre de consolider vos connaissances nouvellement acquises. Nous vous souhaitons comme toujours beaucoup de succès.

EXERCICES

1. Essayez sans les entrer dans l'ordinateur de dire quelle sera la différence sur l'écran entre les résultats des deux instructions suivantes:

   PRINT SPC(5)"TEST1"TAB(15)"TEST2)

   PRINT TAB(5)"TEST1"TAB(15)"TEST2)

   a) La seule différence sera qu'avec la première instruction, TEST1 sera sorti un caractère plus à droite.

   b) Avec la première instruction, 5 espaces sont produits avant que ne commence la sortie du premier mot. La sortie du second mot se fait après 15 autres espaces. Avec la seconde instruction, 5 espaces sont également produits avant que ne commence la sortie du premier mot mais la sortie du second mot se fait après 10 autres espaces seulement.

   c) Avec la première instruction, 5 espaces sont produits avant que ne commence la sortie du premier mot et la sortie du second mot se fait déjà après 10 autres espaces. Avec la seconde instruction, 5 espaces sont également produits avant que ne commence la sortie du premier mot mais la sortie du second mot se fait seulement après 15 autres espaces.

2. Quelle expression obtient-on pour B$ en entrant la suite d'instructions suivantes:

   A$="MOULIN A VENT" (ENTER)

B$=MID$(A$,1,1)+MID$(A$,8,1)+MID$(A$,4,3) (ENTER)

3. Quelle expression obtient-on pour A$ en entrant la suite d'instructions suivantes:

   A$="ROTOR" (ENTER)

   A$=LEFT$(A$,3)+RIGHT$(A$,2) (ENTER)

4. Quelle suite d'instructions permettra de créer à partir de A$="CHAPITEAU DE CIRQUE", B$="CHATEAU DE PITRE"

SOLUTIONS

1. La réponse a) est la bonne.

2. Vous obtenez l'expression MALIN.

3. Vous obtenez à nouveau l'expression ROTOR.

4. B$=LEFT$(A$,3)+MID$(A$,6,8)+MID$(A$,4,3)+MID$(A$,16,1) + RIGHT$(A$,1)

   Il ne s'agit là que d'une possibilité parmi tant d'autres et vous pouvez vous amuser à réaliser cette chaîne de caractères de diverses façons pour maîtriser parfaitement ces instructions de chaînes.

2.6 L'EDITION DES PROGRAMMES

Avant que nous ne passions à la réalisation de programmes de taille plus importante, il nous faut nous intéresser aux instructions qui facilitent le travail de programmation. Le terme d' "édition" des programmes qui nous est plus ou moins imposé par l'origine anglaise du BASIC regroupe tout ce qui se rapporte à la modification d'un programme, qu'il s'agisse d'ajouter, de supprimer ou de modifier des lignes de programme, notamment pour remédier aux erreurs de syntaxe décelées par l'ordinateur.

Nous avons déjà indiqué qu'il est recommandé de numéroter les lignes de programmes de 10 en 10. Vous pouvez pour cela utiliser l'instruction:

AUTO

Si vous entrez AUTO, vous voyez apparaître le nombre 10 et le curseur. L'ordinateur attend ainsi que vous entriez la ligne 10 du programme. Si un programme se trouve déjà en mémoire, l'ordinateur vous signalera les numéros de lignes (multiples de 10) qui correspondent à une ligne du programme en mémoire en plaçant une étoile à la suite de ces numéros de ligne. Si vous voyez donc:

20*

et que vous appuyez directement sur la touche ENTER, la ligne 20 du programme en mémoire ne sera pas modifiée. Si vous commencez au contraire à écrire dans cette ligne, son ancien contenu sera perdu. Vous pouvez mettre fin à la numérotation automatique avec la touche ESC.

Si vous écrivez un programme sans utiliser la numérotation automatique, vous risquez d'avoir besoin à un moment ou un autre de l'instruction:

RENUM

Lorsque vous développez un programme, il arrive fréquemment que vous ayez à rajouter des lignes, de sorte que la numérotation de votre programme se présente vite ainsi:

10
12
15
19
20
21

Ceci peut être très gênant si vous vous apercevez que vous avez maintenant besoin de rajouter une ligne entre 19 et 20 ou entre 20 et 21. Vous pouvez alors utiliser l'instruction RENUM qui renumérotera votre programme de 10 en 10. Dans l'exemple ci-dessus, votre programme se trouvera donc numéroté maintenant de 10 à 60. Cette instruction est très puissante puisqu'elle modifie automatiquement les numéros de ligne placés après les instructions de saut GOTO, GOSUB, etc... en fonction de la nouvelle numérotation.

Vous pouvez également renuméroter uniquement une partie de votre programme:

RENUM X,Y,Z

renumérotera de Z en Z les lignes à partir de la ligne numéro Y en donnant à la ligne numéro Y le numéro X:

RENUM 200,100,5

renumérotera de 5 en 5 les lignes à partir de la ligne numéro 200 en donnant à la ligne numéro 200 le numéro 100.

Il peut arriver que vous ayez besoin de supprimer une partie seulement de votre programme (NEW vous permet en effet de le supprimer entièrement). Vous pouvez alors utiliser l'instruction:

DELETE

dont la syntaxe est:

DELETE 100-200

supprime les lignes 100 à 200,

DELETE -200

supprime toutes les lignes jusqu'à la ligne 200 et

DELETE 200-

supprime toutes les lignes à partir de la ligne 200.

Deux instructions vous permettent de supprimer seulement les variables créées:

CLEAR

supprime toutes les variables et tous les tableaux. Si vous ne connaissez pas encore le concept de "tableau", ne vous inquiétez pas, nous l'expliquerons plus loin. Cette instruction vous permet donc d'annuler en n'importe quel emplacement de votre programme toutes les variables créées.

Vous pouvez aussi ne supprimer qu'un tableau, vous pouvez utiliser l'instruction:

ERASE

ERASE A

supprimera le tableau créé avec DIM A(20). Vous pouvez ainsi libérer de la place en mémoire.

Si vous voulez tester un programme, vous pouvez le faire interrompre en n'importe quel endroit pour vérifier s'il a fonctionné correctement jusqu'ici en plaçant à cet endroit l'instruction:

STOP

Lorsque le programme rencontrera cette instruction, il s'interrompra en affichant le message:

Break in (numéro de ligne)

Vous pouvez alors faire repartir le programme à partir de cet endroit avec l'instruction:

CONT

Il est impossible de développer des programmes d'une certaine taille sans commettre des erreurs de syntaxe (BASIC) ou même des fautes de frappe. Dans ce cas vous pouvez éviter de retaper intégralement la ligne corrigée grâce à l'instruction:

EDIT (numéro de ligne)

Cette instruction affiche en effet la ligne que vous voulez modifier et vous pouvez la corriger comme si vous veniez de l'écrire.

Il est parfois très intéressant de pouvoir suivre le déroulement du programme en connaissant au fur et à mesure les numéros des lignes que le programme est en train d'exécuter. Ceci permet par exemple de faire des comparaisons entre le fonctionnement du programme et votre organigramme. Vous disposez pour ce faire de l'instruction:

TRON

Une fois entrée, cette instruction (TRace ON) affiche pendant le déroulement du programme le numéro de chaque ligne avant de l'exécuter. L'instruction:

TROFF

(TRace OFF) met fin à cette fonction.

L'instruction:

NEW

supprime le programme se trouvant en mémoire. Vous pouvez également avoir recours à l'utilisation simultanée des trois touches SHIFT, CTRL et ESC dont l'effet est encore plus radical puisqu'un RESET est effectué qui place l'ordinateur à nouveau dans l'état où il se trouve après sa mise sous tension (allumage). Toutes les informations en mémoire sont ainsi détruites.

L'instruction:

LIST

vous permet de sortir sur l'écran tout ou partie du programme. Pour définir les lignes que vous voulez faire sortir sur l'écran, vous pouvez employer exactement la même syntaxe que celle de l'instruction DELETE.

Lors du développement de vos programmes, vous pouvez également avoir besoin de deux instructions qui n'ont pas de rapport avec l'édition proprement dite des programmes:

PRINT HIMEM

affiche l'adresse de la dernière case mémoire utilisable par le BASIC. Vous devez normalement obtenir 43903. Vous pouvez modifier cette valeur standard avec l'instruction:

MEMORY (adresse)

qui déplace vers le bas la limite supérieure de la mémoire BASIC (reportez-vous à votre manuel pour plus de précisions). Ceci vous permet par exemple de réserver de la place en mémoire pour des programmes en langage-machine.

Nous en avons fini avec ce chapitre d'introduction à la programmation en BASIC. Le chapitre suivant abordera des structures de programme plus complexes ainsi que la programmation des boucles.

3. STRUCTURES DE PROGRAMMES PLUS COMPLEXES

Nous n'avons abordé jusqu'ici que les programmes à déroulement linéaire. C'est maintenant aux sauts ou aux branchements à l'intérieur d'un programme que nous allons maintenant nous attaquer. L'inconvénient des programmes à déroulement linéaire réside en effet dans le fait que le programme est exécuté une fois de la première ligne à la dernière et qu'il faut ensuite le relancer avec l'instruction RUN si on souhaite continuer à l'utiliser. D'autre part, comme il n'y a pas de branchements à l'intérieur du programme, on est obligé de faire exécuter chaque fois l'intégralité du programme, même si dans certains cas on souhaiterait pouvoir utiliser plusieurs fois de suite telle ou telle partie du programme. Si tous les programmes étaient donc à déroulement linéaire, l'informatique ne pourrait donc résoudre que des problèmes très simples, tels que ceux que nous avons jusqu'ici résolus.

3.1 SAUTS INCONDITIONNELS

La forme la plus simple de branchement à l'intérieur d'un programme, est le saut ou branchement inconditionnel avec l'instruction GOTO que nous allons illustrer à l'aide d'un programme d'exemple. Cette instruction indique à l'ordinateur d'interrompre l'exécution linéaire (ligne par ligne dans l'ordre croissant des numéros de ligne) du programme pour sauter à la ligne dont le numéro est placé à la suite de l'instruction GOTO. Il s'agit d'une instruction de saut inconditionnel car le programme exécute dans tous les cas le branchement voulu par l'instruction GOTO.

Nous allons utiliser pour notre exemple une variable intéressante du système de votre CPC, la variable TIME qui contient en permanence le temps qui s'est écoulé depuis la mise sous tension (allumage) de votre ordinateur. Ce temps est exprimé en 300èmes de seconde. L'instruction:

```
PRINT INT(TIME/300)
```

vous permet donc de savoir à tout moment combien de secondes se sont écoulées depuis que vous avez allumé votre ordinateur. Venons-en maintenant à notre programme d'exemple: entrez d'abord l'instruction NEW puis entrez le programme suivant:

```
10 CLS
20 PRINT CHR$(30);INT(TIME/300)
30 GOTO 20
40 END
```

Si vous lancez ce programme, vous verrez affiché en permanence dans l'angle supérieur gauche de l'écran le nombre de secondes écoulées depuis la mise sous tension de l'ordinateur. L'instruction PRINT CHR$(30) a pour effet de replacer le curseur dans l'angle supérieur gauche de l'écran. Sur d'autres ordinateurs, c'est la touche appelée HOME qui remplit la même fonction.

Vous allez maintenant constater que vous ne pouvez plus arrêter le programme qu'en utilisant à deux reprises la touche ESC. L'inconvénient des branchements ou sauts inconditionnels est en effet qu'ils ne permettent de réaliser que des "boucles infinies": une fois le programme lancé, il ne s'arrêtera plus à moins que vous n'éteigniez votre ordinateur ou que vous n'utilisiez la touche ESC pour l'interrompre.

Nous allons maintenant utiliser l'instruction GOTO pour développer un des exemples que nous avons déjà donnés. Souvenez-vous de l'exercice dans lequel nous vous demandions de réaliser un programme calculant le poids idéal d'une personne.

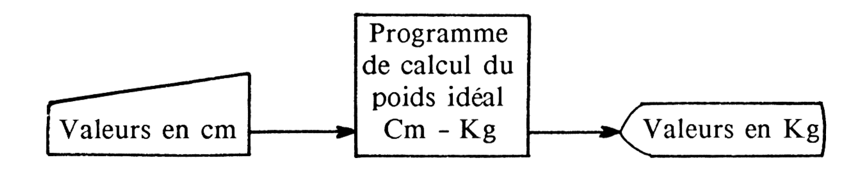
Essayons maintenant de développer ce programme de façon à en faire un petit gag pour vos petites soirées: chacun de vos invités pourra connaître son poids idéal. Sans l'instruction GOTO, ce programme devrait être relancé pour chaque invité. C'est pourquoi nous allons ajouter avant la dernière ligne de la première version de notre programme une instruction GOTO qui imposera à l'ordinateur de sauter à nouveau en début de programme après avoir donné le poids idéal d'un invité:

```
10 INPUT"ENTREE TAILLE EN CM";CM
20 REM CALCUL POIDS IDEAL
30 IDEAL=CM-100
40 REM CALCUL 10 POUR CENT
50 POURCENT=IDEAL/100*10
60 IDEAL=IDEAL-POURCENT
70 PRINT"VOTRE POIDS IDEAL EST DE";IDEAL;"KG"
80 REM SAUT INCONDITIONNEL AVEC GOTO
90 GOTO 10
100 END
```

Vous voyez que le programme, après avoir sorti le poids idéal d'une personne en ligne 70, rencontre en ligne 90 une instruction de branchement inconditionnel. Il saute alors en ligne 10 pour permettre l'entrée des paramètres correspondant à une autre personne. Si vous voulez interrompre ce programme vous n'avez toujours pas d'autre possibilité que d'utiliser la touche ESC. La ligne 100 END est d'ailleurs parfaitement inutile car elle ne pourra jamais être atteinte par le programme puisqu'elle est "court-circuitée" par le saut en ligne 90.

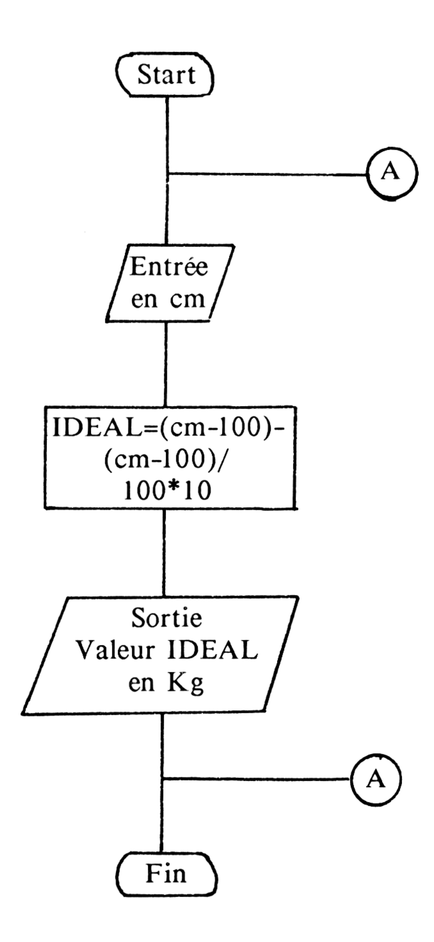
Le plan de flux des données n'est pas affecté par cette nouvelle instruction; il se présenterait de toute façon ainsi:

PLAN DE FLUX DE DONNEES
POUR LE CALCUL DU POIDS IDEAL



Par contre l'organigramme de notre programme est modifié ainsi que vous pouvez le voir page suivante.

ORGANIGRAMME DU PROGRAMME POIDS IDEAL



Vous voyez que nous avons utilisé un nouveau symbole, le connecteur qui permet d'éviter de tracer de grands traits à travers la page pour marquer les sauts du programme. Un premier connecteur figure avant la fin du programme pour indiquer qu'il y a là un saut du programme (GOTO), un second se trouve après le début du programme pour indiquer où aboutit ce saut du programme. les deux connecteurs sont reliés entre eux par la lettre A qui leur est commune. Nous aurions pu également nous passer ici du symbole de fin qui est court-circuité par le premier connecteur comme la ligne 100 END est court-circuitée par la ligne 90 GOTO 10. Vous voyez donc que l'instruction GOTO ne permet à elle seule que de réaliser des boucles sans fin.

3.2 SAUTS CONDITIONNELS

La puissance des ordinateurs vient de leur faculté d'effectuer des décisions logiques ou des comparaisons. Un ordinateur peut en effet tester si une variable est inférieure ou supérieure à 0 et ensuite effectuer ou ne pas effectuer un branchement suivant le résultat de ce test. L'instruction IF ... THEN permet par exemple d'effectuer de tels branchements conditionnels en fonction du résultat d'une comparaison.

3.2.1 IF...THEN...ELSE

Lorsque l'ordinateur rencontre une instruction IF ... THEN ... ELSE (si ... alors ... sinon), il examine d'abord la condition qui a été placée à la suite de IF. Si cette condition est VRAIE, c'est-à-dire si elle est remplie, l'ordinateur exécute la ou les instructions placées à la suite de THEN.

Si par contre cette condition est FAUSSE, c'est-à-dire si elle n'est pas remplie, l'ordinateur exécute la ou les instructions placées à la suite de ELSE et en tout cas il n'exécute pas les instructions placées à la suite de THEN. Le mot ELSE est facultatif.

Le mot IF peut être suivi d'opérateurs logiques, de chaînes de caractères, de variables, de comparaisons et de nombres ou de combinaisons de ces divers éléments.
Le mot THEN est suivi le plus souvent d'un numéro de ligne à laquelle le programme doit sauter. Il peut être aussi suivi d'affectations de valeurs à des variables ou de sauts à des sous-programmes.
Voici maintenant un exemple simple de l'emploi de l'instruction IF ... THEN:

```
10 INPUT "Entrez un nombre";X
20 IF X<0 THEN 50
30 IF X>0 THEN 70
40 IF X=0 THEN 90
50 PRINT "Ce nombre est plus petit que zéro"
60 GOTO 100
70 PRINT "Ce nombre est plus grand que zéro"
80 GOTO 100
90 PRINT "Ce nombre est égal à zéro"
100 END
```

Ce programme vous permet d'entrer un nombre quelconque, après quoi il vous indique si ce nombre est plus petit, plus grand ou égal à zéro. Bien entendu, vous le savez vous-même et le seul intérêt de ce programme est de vous montrer comment l'ordinateur exécute l'instruction IF ... THEN:
Si vous entrez maintenant un nombre supérieur à 0, l'ordinateur rencontre d'abord la ligne 20. Il teste alors si le nombre que vous avez entré est inférieur à 0.

Comme cette condition n'est pas remplie, il ne saute pas en 50 mais passe à la ligne suivante. Il teste alors si le nombre que vous avez entré est supérieur à 0. Comme cette condition est remplie, il exécute l'instruction placée à la suite de THEN et saute en 70. La ligne 70 affiche alors le message indiquant que le nombre entré est supérieur à 0. L'ordinateur rencontre ensuite en ligne 80 l'instruction de branchement inconditionnel GOTO 100 et il saute à la ligne 100 qui termine le programme. Si vous voulez au contraire que le programme tourne continuellement, il vous suffit de remplacer l'instruction END de la ligne 100 par l'instruction GOTO 10.

Après ce programme d'exemple simple, nous allons aborder maintenant un programme déjà un peu plus compliqué. Vous connaissez certainement le jeu où quelqu'un doit penser à un nombre qu'une autre personne doit essayer de deviner. Après chaque question de la personne qui doit deviner, on doit dire si le nombre proposé est plus grand, plus petit ou égal au nombre à découvrir. Nous allons maintenant réaliser ce jeu sur notre ordinateur. Entrez donc le programme suivant:

```
10 REM DEVINER UN NOMBRE
20 CLS:PRINT
30 PRINT "Entrez deux nombres correspondant ";CHR$(10)
40 PRINT "A la limite inférieure et à la limite supérieure.";CHR$(10)
50 INPUT"Limite inférieure";inf
60 INPUT"Limite supérieure";sup
70 Z=INT(sup*RND(1))+inf
80 INPUT"PROPOSEZ UN NOMBRE";sz
90 IF sz<z THEN 120
100 IF sz>z THEN 140
110 IF sz=z THEN 160
120 PRINT"Le nombre à trouver est plus grand"
130 GOTO 80
```

```
140 PRINT"Le nombre à trouver est plus petit"
150 GOTO 80
160 PRINT"Bravo! vous avez trouvé!"
170 PRINT"Voulez-vous rejouer oui/non"
180 INPUT A$
190 IF A$="oui" THEN 20
200 END
```

Les premières lignes n'ont pas besoin de commentaire. Une petite remarque sur la ligne 30: CHR$(10) déplace le curseur d'une ligne vers le bas.

Les lignes 30 à 60 demandent au joueur d'entrer les limites entre lesquelles doit être compris le nombre à deviner. La ligne 70 détermine dans les limites qui viennent d'être définies le nombre à trouver au moyen de la fonction RND. Si vous ne comprenez pas bien la logique de la ligne 70, reportez-vous au chapitre où nous avons traité des nombres aléatoires. La ligne 80 vous demande maintenant d'entrer un nombre. Ce nombre est maintenant comparé en lignes 90 à 110 au nombre qui a été tiré au sort et que contient la variable Z. Suivant que le nombre que vous avez entré est plus grand, plus petit ou égal au nombre à trouver, le programme saute à la ligne correspondante et le programme continue. Si vous avez trouvé le nombre à découvrir, le programme saute en ligne 160 et on vous demande en ligne 170 si vous voulez rejouer. Si vous entrez Oui, la condition posée en ligne 190 se trouve remplie et le programme recommence, sinon le programme est terminé.

L'instruction IF ... THEN en lignes 90 à 110 sert donc à comparer le nombre entré par le joueur (SZ) et le nombre tiré au sort par l'ordinateur (Z). En ligne 190 par contre, l'instruction IF ... THEN sert à comparer une variable alphanumérique. Notez bien que, pour que la condition en ligne 190 soit vraie, il faut absolument que les deux chaînes comparées soient rigoureusement identiques.

Bien entendu, si vous entrez 'yes', le programme sera terminé malgré tout. Si vous voulez rejouer, vous devez absolument entrer 'oui'.

L'instruction IF ... THEN nous permet maintenant également de réaliser des "boucles dirigées", c'est-à-dire non plus simplement des boucles sans fin que le programme parcourt indéfiniment mais des boucles que le programme parcourra jusqu'à ce qu'une condition déterminée ait été remplie. Le petit programme d'exemple suivant vous montre comment programmer une boucle dirigée. Notre programme sort sur l'écran la table de 3:

```
10 A=3
20 PRINT A
30 A=A+3
40 IF A>30 THEN 60
50 GOTO 20
60 END
```

La ligne 10 initialise la variable A avec la valeur 3. La ligne 30 crée ce qu'on appelle un compteur qui ajoute toujours 3 au contenu actuel de la variable A. La ligne 40 teste si A a déjà dépassé 30. Tant que A est inférieur ou égal à 30, le programme continue avec l'instruction GOTO en ligne 50. Nous avons ainsi réalisé une boucle qui sera parcourue exactement 10 fois par le programme. Nous disposons ainsi d'un moyen de créer des boucles qui ne soient parcourues qu'un nombre déterminé de fois.

L'exemple ci-dessus se prête bien à l'emploi du mot ELSE qui va nous permettre ici de faire l'économie de la ligne 50:

```
10 A=3
20 PRINT A
30 A=A+3
40 IF A>30 THEN 60 ELSE 20
60 END
```

Si vous lancez cette nouvelle version du programme, vous constaterez que vous obtenez exactement le même résultat qu'avec la première version: tant que, lors des 10 premiers parcours, A est inférieur à 30 et que la condition après IF est fausse, c'est l'instruction placée après le mot ELSE qui est exécutée. Le programme saute alors à la ligne 20. Dès que A a atteint la valeur 33, le programme est terminé en ligne 60. Nous avons laissé à la ligne 60 son numéro pour bien mettre en évidence la suppression de la ligne 50. Le mot ELSE vous permet donc d'améliorer la présentation de vos programmes et d'économiser des lignes de programme.

Voici maintenant à nouveau quelques exercices pour consolider vos connaissances. Bonne chance!

EXERCICES

1. Ecrivez un programme (théorique) qui calcule suivant votre revenu annuel un impôt de 33 ou de 51%. La limite sera de 50 000F: les revenus supérieurs à 50 000F seront imposés suivant un taux de 51%, les revenus inférieurs de 33%. Le résultat doit être présenté par un texte d'accompagnement.

2. Ecrivez un programme calculant la somme des nombres de 1 à 100.

3. Ecrivez un programme qui sorte 6 nombres aléatoires compris entre 1 et 49.

4. Quels nombres seront sortis par ce programme? Essayez bien sûr de répondre avant d'entrer ce programme.

```
10 A=7
20 A=A+5:Z=Z+1
30 IF Z<9 THEN 20
40 PRINT A,Z
50 END
```

5. Ecrivez un programme qui vous permette de faire rechercher une sous-chaîne quelconque dans une chaîne de caractères quelconque. Testez votre programme en entrant INFORMATIQUE comme chaîne de caractères et FORMAT comme sous-chaîne à rechercher. La difficulté particulière de cet exercice résidera dans le fait que vous devrez résoudre cet exercice sans utiliser l'instruction INSTR. Il s'agira donc pour vous d'écrire une routine en BASIC qui remplace l'instruction INSTR.

SOLUTIONS

1.
```
10 REM ENTREE REVENU ANNUEL
20 INPUT"REVENU ANNUEL EN FRANCS";revenu
30 IF revenu>50000 THEN 70
40 REM CALCUL DE 33 POUR CENT
50 impot=revenu/100*33
60 GOTO 90
70 REM CALCUL DE 51 POUR CENT
80 impot=revenu/100*51
90 PRINT"IMPOT A PAYER ";
100 PRINT impot;" F"
110 END
```

La ligne 20 demande le revenu annuel. La valeur entrée est affectée à la variable revenu. La ligne 30 teste si le revenu est supérieur à 50 000F. Si ce n'est pas le cas, le programme calcule l'impôt sur la base de 33% du revenu et sort l'impôt calculé. Si par contre le revenu est supérieur à 50 000F, la ligne 80 calcule l'impôt sur la base de 51% du revenu et sort l'impôt calculé.

2. Cet exercice peut être résolu d'au moins deux façons différentes. Voyons tout d'abord une première solution avec l'instruction IF ... THEN:

```
10 REM SOMME DES NOMBRES DE 1 A 100
20 A=A+1
30 S=S+A
40 IF A < 100 THEN 20
50 PRINT"LA SOMME DES NOMBRES DE 1 A 100 =";S
60 END
```

La ligne 20 crée un compteur pour les nombres à additionner de 1 à 100. La ligne 30 calcule la somme des différentes valeurs que prend A: 1+2+3+ ... +100. La ligne 40 effectue une comparaison et la ligne 50 sort enfin la somme S des nombres de 1 à 100.
La seconde solution est tirée de la constatation du fait que nous avons ici affaire à une série arithmétique, c'est-à-dire que la différence entre les différents termes de la série est constante. La somme des différents termes de la série peut donc être calculée d'après la formule:

Sn=n/2(A1+An)

n est le nombre de termes de la série pris en compte, A1 est le premier terme de la série et An le dernier terme. La deuxième solution vers laquelle nous nous acheminons ainsi constituerait donc même une solution générale de ce type de problèmes. Voici comment pourrait se présenter notre programme:

```
10 INPUT"Nombre de termes";n
20 INPUT"Premier terme";A1
30 INPUT"Dernier terme"AN
40 REM CALCUL
50 SN=N/2*(A1+AN)
60 REM SORTIE
70 PRINT"La somme est";SN
80 END
```

3.

```
10 REM 6 PARMI 49
20 Z=Z+1
30 L=INT(49*RND(1))+1
40 IF Z > 6 THEN END
50 PRINT L;
60 GOTO 20
```

Pour la première fois, nous n'avons pas placé l'instruction END à la fin (physique) d'un programme. Cette instruction marque en effet la fin logique d'un programme qui ne co\ncide plus nécessairement avec la dernière ligne du programme, dès lors qu'on n'est pas dans un programme à déroulement linéaire. La structure très simple de ce programme n'appelle pas d'autre commentaire.

4. Il fallait comprendre dans cet exercice que seules les dernières valeurs de A et Z sont sorties. L'instruction PRINT est en effet placée en dehors de la boucle. La solution est donc:

52 9

Si vous aviez pensé que la deuxième valeur devait être 8, réfléchissez au fait que le programme saute en ligne 20 tant que Z est inférieur à 9. Ce n'est qu'à partir du moment où Z est égal à 9 que la condition n'est plus remplie et que la programme passe à la sortie en ligne 40.

5.

```
10 REM ENTREE DE CHAINE ET SOUS-CHAINE
20 INPUT"Quelle chaine";A$
30 INPUT"Quelle sous-chaine";B$
40 I=I+1
50 C$=MID$(A$,I,LEN(B$))
60 IF C$=B$ THEN PRINT"TROUVE":END
70 IF I>LEN(A$) THEN PRINT"PAS TROUVE":END
80 GOTO 40
```

Reconnaissons que cet exercice était vraiment difficile puisqu'il s'agissait de simuler l'instruction INSTR. Votre programme ne doit pas nécessairement être identique à celui que nous vous indiquons ici mais il devrait de toute façon contenir l'instruction de création de la chaîne de comparaison car c'était en effet le noeud

du problème. La difficulté était en effet d'indiquer à l'ordinateur avec l'instruction MID$ à partir de quel caractère de la chaîne principale, il devait commencer la comparaison avec la sous-chaîne mais aussi sur combien de caractères devait se faire cette comparaison. Comme la sous-chaîne devait être recherchée dans la totalité du texte de la chaîne principale, nous avons donc réalisé avec I un compteur qui nous permet de commencer la comparaison à partir du premier caractère puis de la poursuivre à partir des caractères suivants tant que la sous-chaîne n'a pas été trouvée. Le programme parcourt une boucle de comparaison tant que I est inférieur à la longueur de A$ qui est fournie par LEN(A$). Pour indiquer au programme combien de caractères de A$ doivent être comparés à la sous-chaîne B$ avec l'instruction MID$, nous utilisons encore une fois la fonction LEN(B$) qui fournit ici le nombre de caractères dont se compose B$. La ligne 60 compare la chaîne à rechercher B$ à la chaîne provisoire C$. Si les deux chaînes sont identiques, le programme est terminé. La ligne 70 demande si toute la longueur de A$ a déjà été examinée. Dans ce cas, cela veut dire que B$ ne peut plus être trouvé. Voici une description du fonctionnement du programme avec un exemple concret:

Rechercher dans la chaîne de caractères A$="INFORMATIQUE"
la chaîne de caractères B$="FORMAT".
Nombre de caractères de B$=6
Le programme crée donc les sous-chaînes provisoires suivantes:

1. INFORM
2. NFORMA
3. FORMAT

La troisième sous-chaîne ainsi créée est la chaîne recherchée. Ouf! C'était un sacré morceau! Nous vous invitons quand même à bien étudier ce programme jusqu'à ce que vous soyez sûr de l'avoir compris dans ses moindres détails. Lorsque ce sera le cas, vous pourrez vous attaquer hardiment au chapitre suivant.

3.2.2 FOR ... TO ... NEXT

Nous avons déjà créé des boucles, au moyen de l'instruction IF ... THEN. Le principe consistait à créer un compteur dont la valeur était augmentée ou diminuée lors de chaque parcours de la boucle. On testait à certains endroits du programme la valeur atteinte par le compteur, et suivant le résultat négatif ou positif du test, le programme sautait à nouveau au début de la boucle ou bien sortait de la boucle. La programmation d'une boucle telle que nous l'avons vue jusqu'ici est donc relativement complexe alors que les boucles sont un dispositif important dans tout programme qui ne se cantonne pas dans les limites du déroulement linéaire. Vous devez donc certainement vous douter déjà que le BASIC doit offrir une solution plus pratique pour arriver au même résultat. Et en effet l'instruction FOR ... TO ... NEXT permet de programmer les boucles beaucoup plus aisément comme le montre ce petit programme d'exemple:

```
10 REM SORTIE DES DIX PREMIERS CARRES
20 CLS:PRINT
30 PRINT"Sortie des 10 premiers carres";CHR$(10)
40 FOR I=1 TO 10
50 PRINT"Le carré de";I;"est";I*I
60 NEXT I
70 PRINT"Fin"
```

Entrez ce programme et lancez-le. L'instruction FOR I=1 crée un compteur I auquel elle affecte la valeur de départ 1. Cette valeur de départ sera augmentée (incrémentée) de 1, jusqu'à ce qu'elle dépasse la valeur finale placée après TO. Ici, 10. Toutes les instructions placées entre FOR ... TO ... et l'instruction NEXT qui ferme la boucle seront exécutées, tant que la valeur du compteur I ne dépassera pas la valeur finale. Quelques exemples:

```
10 A=10:B=20
20 FOR Z=A TO B
30 PRINT Z;
40 NEXT Z
50 END
```

La ligne 10 initialise ici d'abord les variables A et B. La ligne 20 crée maintenant une boucle en utilisant ces deux variables comme valeur de départ et valeur finale. La ligne 30 sort les différentes valeurs de Z, tant que le compteur Z n'est pas supérieur à 20. Vous pouvez d'ailleurs vérifier en mode direct qu'une fois le programme terminé, Z aura bien une valeur supérieure à 20:

```
PRINT Z
```

donnera: 21. Après avoir utilisé des variables pour définir la boucle, voici maintenant un exemple avec des expressions arithmétiques:

```
10 A=10:B=15:C=5
20 FOR Z=A TO A+B-C
30 PRINT Z;
40 NEXT Z
50 END
```

Ce programme revient exactement au même que l'exemple précédent mais la valeur finale est calculée à partir de l'expression A+B-C.

Si vous voulez que la boucle ait un autre pas d'incrémentation que 1, il faut définir ce pas avec l'instruction STEP. Voici par exemple un programme qui sort tous les nombres pairs entre 2 et 20:

```
10 REM NOMBRES PAIRS DE 2 A 20
20 FOR I=2 TO 20 STEP 2
30 PRINT I
40 NEXT I
50 END
```

La valeur de départ, la valeur finale ainsi que le pas d'incrémentation (ou de décrémentation) peuvent être également des nombres négatifs ou des fractions numériques. Voici par exemple un programme de compte à rebours:

```
10 REM COMPTE A REBOURS
20 FOR I=20 TO 0 STEP -1
30 PRINT I
40 NEXT I
50 END
```

Si vous lancez ce programme, vous allez constater que les chiffres défilent très rapidement alors qu'un vrai compte à rebours devrait compter les secondes. Nous allons pour rendre effective la fonction de compte à rebours de notre programme, imbriquer des boucles:

```
10 REM COMPTE A REBOURS
20 FOR I=20 TO 0 STEP -1      ----------------+
30 PRINT I                                    +
40 FOR Z=0 TO 1000      ----------------+     +
50 REM BOUCLE DE TEMPORISATION +              +
60 NEXT Z            ------------------+      +
70 NEXT I                  -------------------+
80 END
```

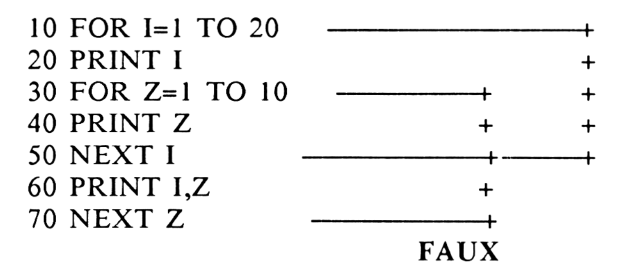
Entrez ce programme (sans les caractères graphiques que nous avons ajouté pour marquer les boucles) et lancez-le: vous voyez que le compte à rebours marque presque exactement les secondes.

Nous avons pour arriver à ce résultat utilisé une boucle de temporisation en lignes 40 à 60. Les boucles de temporisation sont utilisées très souvent, par exemple pour permettre à l'utilisateur de lire un texte avant que le programme n'affiche la page suivante sur l'écran. Examinons le fonctionnement des boucles imbriquées:

La première boucle est ouverte en ligne 20 avec I=20. La ligne 30 sort la valeur actuelle du compteur de la première boucle I. La ligne 40 ouvre la seconde boucle dont le NEXT est en ligne 60. Cette boucle est d'abord parcourue entièrement, c'est-à-dire jusqu'à ce que Z soit supérieur à la valeur finale 1000, avant que la première boucle ne termine son premiers parcours et ne puisse commencer son deuxième.

Les boucles imbriquées augmentent considérablement la puissance de vos programmes mais vous devez faire attention à la manipuler correctement et à ne pas réaliser de boucles croisées. Il doit toujours y avoir une hiérarchie précise entre les différentes boucles imbriquées et la première boucle ouverte doit toujours être la dernière fermée et la dernière ouverte doit toujours être la première fermée. Voici par exemple ce qu'il ne faut jamais faire:

FAUX



```
10 FOR I=1 TO 20
20 PRINT I
30 FOR Z=1 TO 10
40 PRINT Z
50 NEXT I
60 PRINT I,Z
70 NEXT Z
```

FAUX

Si vous avez imbriqué plusieurs boucles et que vous voulez les fermer l'une après l'autre sans intercaler d'instructions entre les fins des différentes boucles, vous ne devez pas obligatoirement utiliser un NEXT spécial pour chaque instruction. Il suffit d'une seule instruction NEXT suivie des noms des différents compteurs de boucle, placés dans le bon ordre et séparés entre eux par une virgule:

```
10 FOR I=1 TO 10
20 FOR Z=1 TO 10
30 PRINT I;Z
40 NEXT Z,I
50 END
```

Vous voyez que nous avons fermé deux boucles avec une seule instruction NEXT mais que nous avons aussi respecté la règle qui veut que la dernière boucle ouverte (Z) soit fermée avant la première boucle ouverte (I).

Une autre faute à éviter lorsqu'on utilise des boucles dans un programme est de sauter à l'intérieur d'une boucle. Il est en effet fréquent qu'une boucle se compose de plusieurs lignes de programme et il est tout à fait possible de sauter directement à une de ces lignes. Mais lorsque l'ordinateur rencontrera l'instruction NEXT qui ferme la boucle, il affichera le message d'erreur:

Unexpected NEXT in (numéro de ligne)

Si le programme n'est pas passé par l'instruction FOR qui ouvrait la boucle, il ne peut en effet pas savoir où il doit retourner quand il rencontre ce NEXT. Il s'agit d'ailleurs au fond de la même faute que celle qui consiste à réaliser des boucles croisées. Il faut toujours se rappeler qu'une boucle doit être parcourue entièrement, ou ne pas être parcourue.

C'est là que se manifeste l'intérêt des organigrammes qui doivent normalement permettre d'éviter que de telles erreurs se produisent en les rendant beaucoup plus tangibles et évidentes.

Il faut d'autre part penser, si la valeur de départ que vous définissez est supérieure à la valeur finale, que vous devez indiquer un pas d'incrémentation négative (=décrémentation). Sinon la boucle ne sera pas parcourue une seule fois:

```
10 FOR A=5 TO 1
20 PRINT A
30 NEXT A
40 END
```

On a ici oublié en ligne 10 d'indiquer le pas avec STEP -1. La valeur A ne sera donc pas sortie une seule fois puisque A est dès le départ supérieure à la valeur finale 1.

Le programme ci-dessus est l'exemple d'une erreur de programmation mais vous pouvez utiliser le même principe pour provoquer un arrêt anticipé d'une boucle en affectant au compteur pendant le parcours d'une boucle une valeur supérieure à la valeur finale, si certaines conditions se sont produites qui rendent inutiles d'autres parcours. Vous pouvez également utiliser comme paramètres de l'instruction FOR ... TO des variables de façon à pouvoir obtenir des boucles de différentes longueurs. Voici d'abord un exemple d'interruption anticipée d'une boucle:

```
10 REM AUGMENTATION BRUSQUE DE LA VALEUR
DU COMPTEUR
20 FOR A=0 TO 20
30 PRINT A
40 IF A=12 THEN A=20
50 NEXT A
60 END
```

Vous voyez que cette boucle qui devait normalement sortir les nombres de 0 à 20 ne sort que les nombres de 0 à 12 à cause de ligne 40 qui met fin à la boucle dès que A vaut 12 en affectant 20 à la variable de compteur A. Retenez cette possibilité qui peut vous être utile un jour, bien qu'elle ne soit pas très souvent employée. Par contre, la possibilité suivante qui consiste à créer des boucles à "géométrie variable" est très utilisée, notamment dans les programmes de gestion de fichier pour rechercher par exemple certaines combinaisons de caractères:

```
10 INPUT"ENTREZ UN MOT";A$
20 FOR A=1 TO LEN(A$)
30 PRINT LEFT$(A$,A)
40 NEXT A
50 FOR A=LEN(A$) TO 1 STEP -1
60 PRINT RIGHT$(A$,A)
70 NEXT A
80 END
```

Entrez ce programme, lancez-le et entrez votre nom. Vous voyez que c'est ici la longueur du mot entré qui détermine le nombre de parcours de chaque boucle.

Essayons maintenant de résumer tout ce que nous avons appris sur les boucles FOR ... TO ... NEXT:

1. A chaque instruction FOR ... TO ... doit répondre une instruction NEXT. Une seule instruction NEXT peut fermer plusieurs boucles à la fois si elle est suivie des variables-compteurs de ces différentes boucles, placées dans l'ordre inverse de l'ordre d'ouverture des boucles correspondantes et séparées entre elles par une virgule.

2. On ne doit jamais sauter à l'intérieur d'une boucle sous peine de provoquer l'interruption du programme avec sortie d'un message d'erreur.

3. Si la valeur de départ est supérieure à la valeur finale du compteur d'une boucle, il faut indiquer le pas négatif de la boucle (avec STEP -x).

4. Une boucle FOR ... NEXT sera parcourue tant que la valeur de la variable-compteur ne sera pas supérieure à la valeur finale.

Ces règles ne s'appliquent qu'au BASIC du CPC et ne doivent donc pas être généralisées hâtivement. Les différents dialectes BASIC existants divergent en effet quelque peu dans le traitement des boucles FOR ... NEXT.

3.2.3 WHILE ... WEND

La combinaison des instructions WHILE ... WEND vous offre un autre moyen particulièrement puissant et plus souple que FOR ... TO ... NEXT de réaliser des boucles. Vous n'avez pas en effet à définir d'incrément fixe ou de pas définitif.

Vous pouvez ouvrir une boucle avec WHILE que vous devez refermer avec WEND. L'expression logique qui est placée à la suite de WHILE est testée lors de chaque parcours de la boucle. Tant que cette expression est VRAIE, la boucle est parcourue jusqu'à WEND. Si cette expression n'est plus VRAIE, le programme saute à l'exécution des instructions placées après WEND. Vous pouvez par exemple faire interrompre une telle boucle par pur hasard, comme le montre l'exemple suivant:

```
10 WHILE A<100
20 A=INT(101*RND(1))
30 Z=Z+1
40 WEND
50 PRINT A,Z
60 END
```

Cette boucle sera parcourue tant que A sera plus petit que 100. La variable 100 reçoit sa valeur de la fonction aléatoire RND. Si A reçoit la valeur 100, le programme saute en ligne 50 et sort les valeurs de A et de Z. Le compteur Z vous indique ainsi combien de fois la boucle a été parcourue.

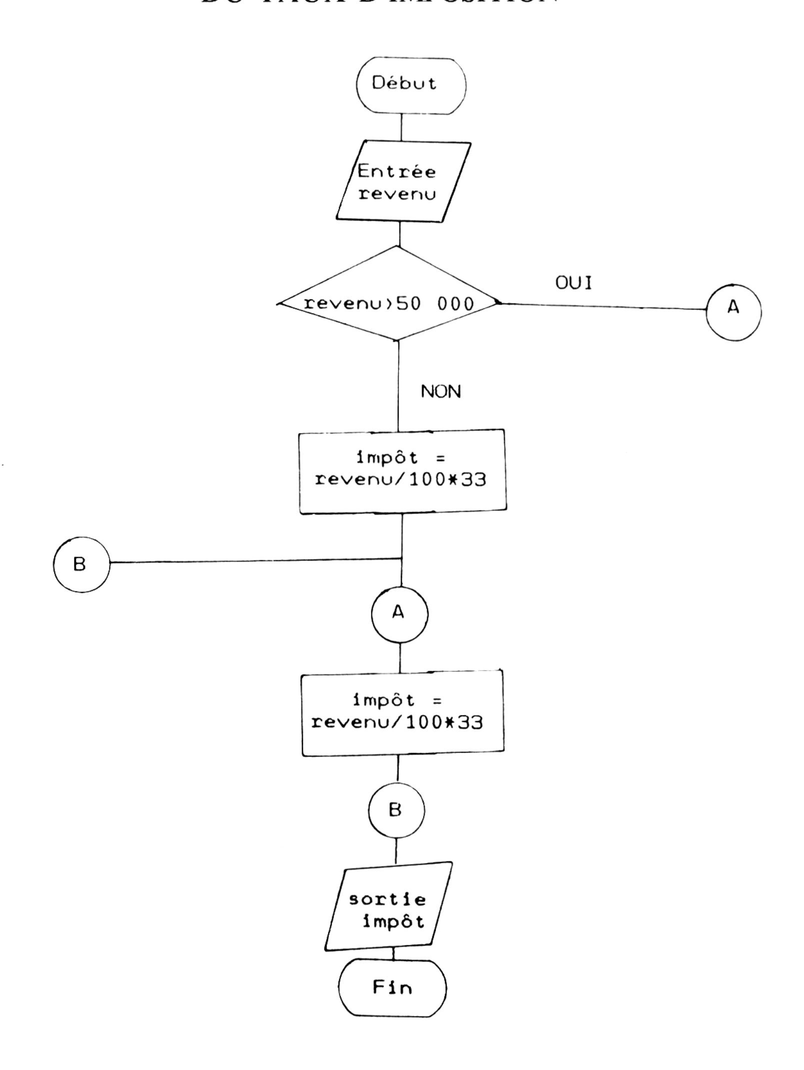
Voici maintenant les règles générales qu'on peut définir pour l'utilisation des boucles WHILE ... WEND:

1. Si vous connaissez d'avance le nombre de parcours de boucle nécessaires, il vaut mieux utiliser la boucle FOR ... TO ... NEXT.

2. Si vous ne connaissez pas d'avance le nombre de parcours de boucle, il vaut mieux utiliser les boucles avec IF ... THEN ou WHILE ... WEND.

Maintenant que nous avons étudié les branchements inconditionnels et les branchements conditionnels ainsi que la technique et les instructions de boucle, il nous manque encore le moyen de représenter ces structures de programmation dans nos organigrammes. C'est le losange qui est le symbole utilisé pour marquer un branchement logique (conditionnel) dans un organigramme. Voyons par exemple comment pourrait se présenter l'organigramme de notre programme de calcul du taux d'imposition:

ORGANIGRAMME DU PROGRAMME DE CALCUL DU TAUX D'IMPOSITION

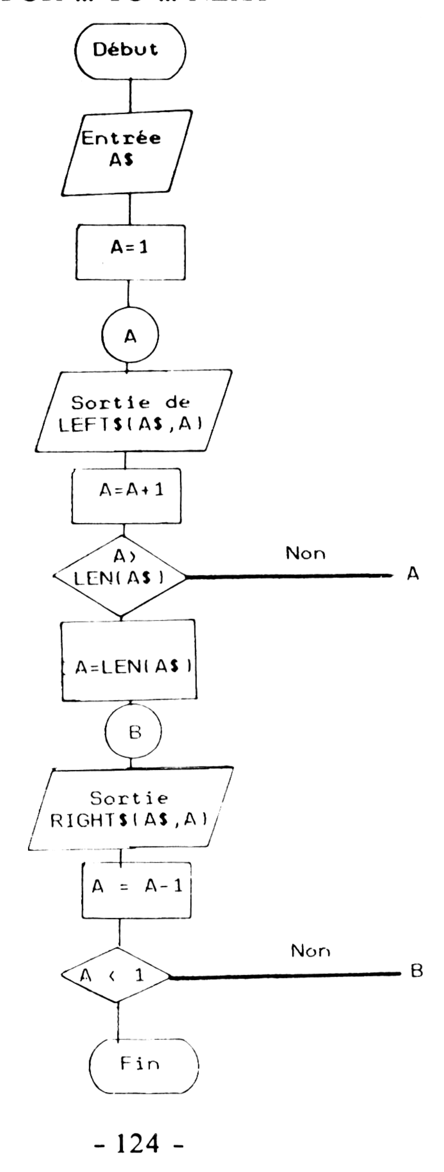


Le losange est donc le symbole d'un branchement logique: il comporte en effet une branche oui et une branche non. Si la condition figurant à l'intérieur du losange est remplie, le programme suit donc la branche oui qui le mène au connecteur A par lequel il saute au deuxième connecteur A. Si la condition n'est pas vérifiée, le programme poursuit son déroulement normal (linéaire) par la branche non. Dans notre exemple le branchement conditionnel se fait donc à travers la branche oui mais un branchement conditionnel peut aussi bien passer par la branche non. Le connecteur de saut A amène le programme au calcul de l'impôt avec un taux de 51 pour cent. L'impôt est ensuite sorti.

Si la condition n'est pas remplie l'impôt est calculé avec un taux de 33 pour cent. Le programme arrive alors au connecteur de saut B qui marque ici un branchement inconditionnel. Il faut veiller à ce que le connecteur de saut B soit bien placé avant le deuxième connecteur A, sinon il y aura une erreur de logique dans notre organigramme.

Cet organigramme nous montre comment représenter l'instruction IF ... THEN dans un organigramme. Il nous reste encore à apprendre comment représenter une boucle FOR ... NEXT dans un organigramme. Vous trouverez à la page suivante l'organigramme du dernier programme d'exemple du chapitre sur les boucles FOR ... TO ... NEXT.

ORGANIGRAMME DU PROGRAMME D'EXEMPLE FOR ... TO ... NEXT



Vous connaissez déjà tous les symboles utilisés dans cet organigramme. Le premier rectangle affecte la valeur 1 comme valeur de départ au compteur A. Ensuite vient la sortie d'une sous-chaîne avec LEFT$(A$,A). Le compteur est alors incrémenté de 1. Le losange demande si le compteur est déjà plus grand que le nombre de caractères qui composent A$. Tant que ce n'est pas le cas, le programme saute à travers le connecteur A au début de la boucle où se trouve l'autre connecteur A. Voici donc comment on représente une boucle dans un organigramme.

Si le compteur est plus grand que LEN(A$), c'est la seconde boucle qui entre en action. La seconde boucle a été représentée de la même manière que la première, si ce n'est que nous avons changé le nom des connecteurs pour les distinguer de ceux de la première boucle.

Vous disposez maintenant de tout l'arsenal de symboles nécessaire pour réaliser des organigrammes pour tout type de programme. Nous en aurons bientôt fini avec l'étude des structures de programmation lorsque nous aurons appris à manier les instructions de sauts calculés. C'est l'objet du chapitre suivant.

3.3 INSTRUCTIONS DE SAUTS CALCULES

Les instructions de sauts calculés permettent d'arriver à un déroulement plus souple du programme alors que nous n'avons vu jusqu'ici que des instructions de saut à une ligne déterminée. Le numéro de ligne d'une instruction GOTO par exemple ne peut pas être influencé d'une façon ou d'une autre: avec GOTO 100, le programme sautera toujours à l'adresse 100. Il serait pourtant très intéressant de pouvoir entrer une valeur au début d'un programme, en fonction de laquelle le programme sauterait à une ligne différente. Ceci peut d'ailleurs être réalisé avec l'instruction IF ... THEN mais que de façon très lourde comme le montre l'exemple suivant:

```
10 REM SAUT A DIFFERENTES LIGNES
20 PRINT"Entrez un nombre entre";
30 PRINT CHR$(10) "1 et 4."
40 PRINT
50 INPUT"Quel nombre";Z
60 IF Z = 1 THEN 100
70 IF Z = 2 THEN 200
80 IF Z = 3 THEN 300
90 IF Z = 4 THEN 400
100 PRINT"Saut en ligne 100"
110 GOTO 410
200 PRINT"Saut en ligne 200"
210 GOTO 410
300 PRINT"Saut en ligne 300"
310 GOTO 410
400 PRINT"Saut en ligne 400"
410 END
```

Ce programme arrive bien au résultat recherché: suivant le nombre entre 1 et 4 que vous entrez, le programme saute à une ligne différente.

Mais vous voyez aussi que nous sommes obligé d'utiliser une ligne différente avec l'instruction IF ... THEN pour chacune des possibilités.
L'instruction ON ... GOTO va nous permettre d'obtenir ce même résultat de façon plus simple et plus rapide:

ON (variable) GOTO (numéro de ligne),(numéro de ligne) ...

Cette extension de l'instruction GOTO permet en effet de réaliser des sauts calculés à différentes lignes de programme, suivant la valeur de la variable placée à la suite de ON. La valeur de cette variable doit être comprise entre 0 et le nombre de numéros de ligne placés à la suite de GOTO. Si cette variable a une valeur non entière, les décimales sont ignorées. Si cette valeur est négative, vous obtenez le message d'erreur:

Improper argument in (numéro de ligne)

Si la variable est nulle ou si elle a une valeur supérieur au nombre de numéros de ligne placés après GOTO, c'est l'instruction qui suit l'instruction ON ... GOTO qui sera exécutée. Voici quelques exemples simples pour bien comprendre le mécanisme de cette instruction:

<u>EXEMPLES:</u>

a)
```
10 ON Z GOTO 100,200,250,300
20 PRINT
.
.
.
```

Si la variable Z vaut 1, le programme saute en ligne 100. Si la valeur Z acquiert par la suite (dans une boucle par exemple) les valeurs 2, 3 puis 4, le programme sautera respectivement en ligne 200, 250 puis 300.

Si Z vaut ensuite 5 ou plus, le programme passera à l'exécution de l'instruction suivante, le PRINT qui figure en ligne 20.

b)
```
10 ON Z+3/4 GOTO 100,200,300
20 PRINT
.
.
.
```

Vous voyez que vous pouvez aussi utiliser une expression arithmétique à la place d'une variable. L'intérêt principal de l'instruction ON ... GOTO est qu'elle remplace plusieurs instructions IF ... THEN, ce qui permet au programmeur de gagner du temps et également d'économiser de la place en mémoire BASIC. Voici par exemple une version améliorée de l'exemple précédent, cette fois avec l'instruction ON ... GOTO:

```
10 REM SAUT A DIFFERENTES LIGNES AVEC ON ... GOTO
20 PRINT"Entrez un nombre entre";
30 PRINT CHR$(10) "1 et 4."
40 PRINT
50 INPUT"Quel nombre";Z
60 ON Z GOTO 100,200,300,400
100 PRINT"Saut en ligne 100"
110 GOTO 410
200 PRINT"Saut en ligne 200"
210 GOTO 410
300 PRINT"Saut en ligne 300"
310 GOTO 410
400 PRINT"Saut en ligne 400"
410 END
```

Vous voyez que nous avons économisé trois lignes de programme. Sur des programmes plus longs, la place ainsi gagnée peut être considérable.

Nous avons d'autre part utilisé dans cet exemple une technique qui peut se révéler très intéressante lorsqu'il s'agit de développer des programmes d'une taille relativement importante. Nous avons en effet volontairement numéroté les différentes lignes auxquelles le programme doit sauter de 100 en 100 alors que ce n'était pas nécessaire a priori. Lorsqu'on réalise l'organigramme d'un tel programme, les différentes lignes sont représentées par des branchements horizontaux. Mais comme on ne peut pas savoir avant d'avoir développé le programme quels numéros de ligne correspondront à ces divers branchements, on choisit volontairement des numéros de ligne avec un écart suffisamment important pour que les différentes parties du programme puissent entrer entre ces numéros de ligne.

Les numéros de ligne figurant dans une instruction ON ... GOTO représentent en effet souvent différentes parties d'un programme dont chacune a une fonction spécifique. Il est donc intéressant de faire commencer chacune de ces parties par un numéro "rond". On peut donc espacer ces numéros de ligne de 100 en 100 comme dans notre exemple ou de 1000 en 1000, etc... Ceci permet de rendre le listing du programme plus lisible.

3.3.1 PROGRAMME D'EXEMPLE "COURS DE CALCUL"

Nous connaissons maintenant un jeu d'instructions suffisamment important pour que nous puissions oser nous attaquer à un projet relativement ambitieux. Supposons donc que vous vouliez réaliser pour vos enfants un cours de calcul avec les quatre opérateurs arithmétiques de base sur le CPC. Ce cours devra avoir les caractéristiques suivantes:

1. Choix d'une des quatres opérations fondamentales ou fin du programme
2. Création d'un exercice qui devra être trouvé en un maximum de trois tentatives

3. Après l'échec de la troisième tentative, le résultat devra être affiché
4. Entrée du plus grand nombre pouvant entrer en jeu dans chaque exercice
5. Demander après chaque exercice, si l'utilisateur veut faire un autre exercice avec le même type d'opérations

Voici maintenant le listing de notre programme de cours de calcul. Comme ce programme est tout de même assez long, voici comment vous pourrez le sauvegarder sur cassette (ou disquette) après l'avoir entré : placez d'abord une cassette (ou disquette) de qualité correcte dans votre lecteur de cassette (ou disquette) et, après avoir entré le programme dans l'ordinateur, entrez l'instruction suivante:

SAVE"CALCUL"

Appuyez ensuite sur la touche ENTER.

Le programme sera alors sauvegardé sur la cassette (ou disquette).

Si vous voulez réutiliser plus tard votre programme, vous n'aurez plus qu'à le recharger de la cassette (ou disquette) dans l'ordinateur. Utilisez pour cela l'instruction LOAD. Vous pouvez également charger votre programme avec l'instruction RUN"CALCUL". Dans ce cas, le programme sera lancé automatiquement après avoir été chargé.
Notez que nous aborderons les instructions de sauvegarde et de chargement plus en détail au chapitre 4.5.

```
5 REM ***** MENU *****
10 CLS:f=0
20 PRINT
30 PRINT TAB(12)"COURS DE CALCUL"
40 PRINT:PRINT
50 PRINT TAB(12)"Choisissez:"
60 PRINT
70 PRINT TAB(12)"1 pour l'addition"
80 PRINT
90 PRINT TAB(12)"2 pour la soustraction"
100 PRINT
110 PRINT TAB(12)"3 pour la division"
120 PRINT
130 PRINT TAB(12)"4 pour la multiplication"
133 PRINT
135 PRINT TAB(12)"5 pour FIN"
140 PRINT
150 PRINT TAB(12);:INPUT "Quel nombre";z
160 IF z<1 OR z>5 THEN 10
170 ON z GOTO 200,600,1000,1300,1600
200 REM **********
210 REM  ADDITION
220 REM **********
230 CLS
240 PRINT TAB(10)"Entrez le plus grand nombre"
250 PRINT
260 PRINT TAB(10)"pour l'addition"
270 PRINT
280 PRINT TAB(10);:INPUT"Plus grand nombre";gr
290 REM
299 REM
300 REM GENERATION DE NOMBRES ALEATOIRES
301 REM
310 a1=INT(gr*RND(1))+1
320 a2=INT(gr*RND(1))+1
329 REM
330 REM CALCUL DU RESULTAT
331 REM
340 er=a1+a2
350 CLS
```

```
360 PRINT
370 PRINT a1 "+" a2 "=";
380 INPUT es
390 IF es=er THEN PRINT:PRINT TAB(10)"C'EST JUSTE!":f=0:GOTO
 450
400 PRINT:PRINT TAB(10)"C'EST FAUX!"
410 FOR i=1 TO 2000:NEXT i
420 f=f+1
430 IF f<=2 THEN 350
440 PRINT
450 FOR i=0 TO 2000:NEXT i
460 PRINT TAB(5) "Le resultat est ";er
470 FOR i=0 TO 3000:NEXT i
480 PRINT TAB(5) "Encore un exercice o/n";
490 INPUT a$
500 IF a$="o" THEN f=0:GOTO 300
510 GOTO 10
600 REM **************
610 REM  SOUSTRACTION
620 REM **************
630 CLS
640 PRINT TAB(10)"Entrez le plus grand nombre"
650 PRINT
660 PRINT TAB(10)"pour la soustraction"
670 PRINT
680 PRINT TAB(10);:INPUT"Plus grand nombre";gr
690 REM
700 REM GENERATION DE NOMBRES ALEATOIRES
701 REM
710 a1=INT(gr*RND(1))+1
720 a2=INT(gr*RND(1))+1
730 REM CALCUL DU RESULTAT
740 IF a1 < a2 THEN i=a1:a1=a2:a2=i
750 CLS
760 PRINT
770 er=a1-a2
780 PRINT a1 "-" a2 "=";
790 INPUT es
800 IF es=er THEN PRINT:PRINT TAB(10)"C'EST JUSTE!":f=0:GOTO
 860
```

```
810 PRINT:PRINT TAB(10)"C'EST FAUX!"
820 FOR i=1 TO 2000:NEXT i
830 f=f+1
840 IF f<=2 THEN 750
850 PRINT
860 FOR i=0 TO 2000:NEXT i
870 PRINT TAB(5) "Le resultat est ";er
880 FOR i=0 TO 3000:NEXT i
890 PRINT TAB(5) "Encore un exercice o/n";
900 INPUT a$
910 IF a$="o" THEN f=0:GOTO 710
920 GOTO 10
1000 REM **********
1001 REM  DIVISION
1002 REM **********
1010 CLS
1020 PRINT TAB(10)"Entrez le plus grand nombre"
1030 PRINT
1040 PRINT TAB(10)"pour la division"
1050 PRINT
1070 PRINT TAB(10);:INPUT"Plus grand nombre";gr
1079 REM
1080 REM GENERATION DE NOMBRES ALEATOIRES
1081 REM
1090 a1=INT(gr*RND(1))+1
1100 a2=INT(gr*RND(1))+1
1109 REM
1110 REM CALCUL DU RESULTAT
1111 REM
1120 er=a1*a2
1130 CLS
1140 PRINT
1150 PRINT er "/" a1 "=";
1160 INPUT es
1170 IF es=a2 THEN PRINT:PRINT TAB(10)"C'EST JUSTE!":f=0:GOT
O 1230
1180 PRINT:PRINT TAB(10)"C'EST FAUX!"
1190 FOR i=1 TO 2000:NEXT i
1200 f=f+1
1210 IF f<=2 THEN 1130
```

```
1220 PRINT
1230 FOR i=0 TO 2000:NEXT i
1240 PRINT TAB(5) "Le resultat est ";a2
1250 FOR i=0 TO 3000:NEXT i
1260 PRINT TAB(5) "Encore un exercice o/n";
1270 INPUT a$
1280 IF a$="o" THEN f=0:GOTO 1090
1290 GOTO 10
1300 REM ****************
1301 REM  MULTIPLICATION
1302 REM ****************
1310 CLS
1320 PRINT TAB(10)"Entrez le plus grand nombre"
1340 PRINT
1350 PRINT TAB(10)"pour la multiplication"
1360 PRINT
1370 PRINT TAB(10);:INPUT"Plus grand nombre";gr
1379 REM
1380 REM GENERATION DE NOMBRES ALEATOIRES
1381 REM
1390 a1=INT(gr*RND(1))+1
1400 a2=INT(gr*RND(1))+1
1409 REM
1410 REM CALCUL DU RESULTAT
1411 REM
1420 er=a1*a2
1430 CLS
1450 PRINT a1 "*" a2 "=";
1460 INPUT es
1470 IF es=er THEN PRINT:PRINT TAB(10)"C'EST JUSTE!":f=0:GOT
O 1530
1480 PRINT:PRINT TAB(10)"C'EST FAUX!"
1490 FOR i=1 TO 2000:NEXT i
1500 f=f+1
1510 IF f<=2 THEN 1430
1520 PRINT
1530 FOR i=0 TO 2000:NEXT i
1540 PRINT TAB(5) "Le resultat est ";er
1550 FOR i=0 TO 3000:NEXT i
1560 PRINT TAB(5) "Encore un exercice o/n";
```

```
1570 INPUT a$
1580 IF a$="o" THEN f=0:GOTO 1390
1590 GOTO 10
1600 CLS
1610 END
```

Examinons maintenant les points les plus importants de ce listing. Les lignes 10 à 150 construisent ce qu'on appelle un menu. Un menu est une partie du programme où l'utilisateur peut choisir entre diverses fonctions du programme. Tout programme qui se respecte doit proposer au moins un menu.

La ligne 150 vous demande d'entrer un nombre correspondant au point du programme que vous choisissez. La ligne 160 teste si le nombre que vous avez entré est valable. C'est un bel exemple d'opération logique: si un nombre entré est SOIT inférieur à 1 SOIT supérieur à 5, le programme saute à la ligne 10. Il suffit qu'une seule de ces conditions soit réalisée pour que le branchement se produise, c'est pourquoi nous avons utilisé l'opérateur logique OR (ou inclusif). La ligne 170 nous montre ensuite un exemple d'emploi de l'instruction ON ... GOTO. Nous avons ici économisé 5 lignes de IF ... THEN.

Les lignes 200 à 220 servent à souligner le début de la partie du programme qui traite l'addition. Une telle présentation est très appréciable pour le programmeur quand il a affaire à un programme assez long.

Elle permet en effet de se repérer plus facilement dans le listing et donc de retrouver plus vite une ligne déterminée du programme.

La ligne 230 vide l'écran. Les lignes suivantes demandent à l'utilisateur d'entrer un nombre qui sera la valeur maximum que pourront atteindre les nombres à additionner. Deux nombres aléatoires A1 et A2 sont ensuite produits, à partir desquels sera créé l'exercice d'addition. La ligne 370 présente l'exercice sur l'écran. La ligne 380 permet à l'utilisateur d'entrer un résultat que la ligne 390 compare alors à la valeur calculée en ligne 340. Si le résultat entré est faux, la ligne 400 sort d'abord une ligne d'espaces sur l'écran puis le message "C'est faux!". La ligne 410 est une boucle de temporisation permettant à l'utilisateur de bien lire le message. La ligne 420 incrémente un compteur qui compte le nombre de réponses fausses. Nous voulions en effet que le résultat correct soit affiché au bout de trois réponses fausses. La ligne 430 teste si le nombre de trois réponses fausses est déjà atteint. Tant que ce n'est pas le cas, le programme retourne en ligne 350 pour présenter l'exercice à nouveau. Si trois réponses fausses ont été données, le programme passe à l'exécution de la suite du programme, en ligne 440. Si une réponse juste a été donnée, le programme saute de la ligne 390 à la ligne 450 qui est une boucle de temporisation. La ligne 460 affiche ensuite le résultat. Après une autre temporisation, le programme demande en ligne 480 si l'utilisateur veut faire un autre exercice. Si l'utilisateur entre ici un "O", le compteur F est d'abord remis à 0 (réinitialisé) et le programme saute en ligne 300 où le programme tire deux nouveaux nombres au sort et propose un nouvel exercice.
Si l'utilisateur entre autre chose que "O", le programme retourne en ligne 10 où le menu est à nouveau présenté.

Le compteur F est remis à 0 (réinitialisé) avant tout nouvel exercice car sinon le test IF F<=2 en ligne 430 serait négatif dès après la première réponse fausse du second exercice. F vaudrait en effet déjà 3.

Il convient donc, lorsqu'on utilise plusieurs fois dans un programme de tels compteurs, de ne jamais oublier de les réinitialiser chaque fois que nécessaire.

Les autres parties du programme, "soustraction", "division" et "multiplication" sont construites fondamentalement de la même façon et ne se distinguent que par les petites adaptations qui ont été rendues nécessaires pour chaque opération. Voici par exemple comment a été réalisée la partie du programme pour la soustraction.

La ligne 740 permet d'empêcher des résultats négatifs. En effet, si A1 est supérieur à A2, l'exercice pourra être proposé sans problème. Si par contre A2 est supérieur à A1, A1 - A2 donnera un résultat négatif, ce que nous voulons éviter. C'est pourquoi nous avons ajouté la ligne 740 qui échange dans ce cas les valeurs des deux variables A1 et A2. Notez à cet égard qu'un échange ne peut être programmé ainsi:

A1=A2:A2=A1   <INCORRECT!>

En effet dans ce cas A1 se verrait d'abord affecter la valeur de A2. A2 se verrait ensuite affecter la nouvelle valeur de A1, c'est-à-dire la valeur de A2. A2 n'aurait donc pas changé de valeur, et A1 serait égal à A2.

Pour échanger entre elles les valeurs de deux variables, il faut donc absolument "sauver" la valeur d'une de ces variables, c'est-à-dire l'affecter à une variable provisoire utilisée uniquement pour l'échange. Ici, la valeur de A1 est sauvée dans la variable I. La valeur de A2 est affectée à A1 et l'ancienne valeur de A1, c'est-à-dire la valeur de I, est affectée à A2. Cette technique d'échange des variables est très importante et vous aurez certainement souvent à y recourir. Assurez-vous donc que vous l'avez bien comprise.

Ce point était la seul différence importante avec l'addition. Nous avons également utilisé pour la division un petit "truc" qui est propre à cette partie du programme. Le but était toujours le même, à savoir s'assurer que le diviseur ne serait pas plus grand que le dividende et que le résultat serait un nombre entier. Nous avons donc en ligne 1120 calculé le résultat en multipliant A1 par A2, comme pour la multiplication. Mais pour l'exercice, c'est le "résultat" ER qui est divisé par A1. Le résultat à trouver est donc A2 qui est toujours un nombre entier.

Il n'y a pas de remarque particulière à faire sur la multiplication.

Voilà ce que nous avions à dire sur ce programme qui vous montre un exemple, très proche de la programmation pratique, d'une utilisation de l'instruction ON ... GOTO. Avant que nous ne vous proposions de tester vos nouvelles connaissances, nous allons aborder une instruction très voisine de l'instruction ON ... GOTO, l'instruction ON ERROR GOTO. Mais résumons auparavant les principaux points qu'il faut retenir pour utiliser ON ... GOTO:

1. La valeur qui suit le mot ON peut être une variable, une constante ou une expression arithmétique. Elle désigne la place du numéro de la ligne à laquelle le programme doit sauter dans la liste de numéros de ligne qui suit le mot GOTO.

2. Si cette valeur est nulle ou si elle est plus grande que le nombre de numéros de ligne suivant GOTO, c'est l'instruction suivant l'instruction ON ... GOTO qui sera exécutée.

3. Les valeurs inférieures à 0 ou supérieures à 255 provoquent une interruption du programme avec sortie du message d'erreur:

Improper argument in (numéro de ligne)

4. ON ... GOTO remplace plusieurs instructions IF ... THEN.

3.3.2 SAUTS AVEC ON ERROR GOTO

Cette variante particulière de l'instruction ON ... GOTO est utilisée pour faire gérer des erreurs qui pourraient se produire par le programme lui-même. L'instruction ON ERROR GOTO est placée le plus souvent au début du programme et l'ordinateur "note" dans ses tablettes à quel endroit du programme cette instruction est apparue pour la première fois. Toutes les erreurs qui se produisent après cette instruction font que le programme saute à la ligne dont le numéro suit l'instruction ON ERROR GOTO. Ce numéro doit correspondre à un petit programme qui réagisse en fonction de l'erreur. Mais pour réaliser ce programme, encore faut-il savoir quelle erreur s'est produite, et en quelle ligne du programme.

Le CPC dispose de deux variables qui peuvent être interrogées à ce sujet:

ERR

et

ERL

Si vous entrez ERR, vous obtenez le numéro de l'erreur qui s'est produite. Vous pouvez alors voir dans l'annexe 8 de votre manuel à quelle erreur correspond ce numéro d'erreur. Vous pouvez donc faire sortir par votre petit programme de traitement des erreurs vos propres messages d'erreur, en fonction de la valeur de ERR et éviter ainsi une interruption inopinée du programme.

La variable ERL vous permet ensuite de demander le numéro de la ligne où l'erreur s'est produite.

Cette méthode de traitement des erreurs vous permet de faire exécuter dans le cours du programme un certain nombre de tâches avant que le programme ne soit interrompu définitivement. Vous pouvez ainsi dans un programme de gestion de fichier par exemple fermer tous les fichiers dès qu'une erreur apparaît, ce qui évitera toute perte de données. Le programme peut alors sortir son propre message d'erreur avant d'être interrompu.

Il est même parfois possible de laisser le programme suivre son cours après l'apparition d'une erreur. Il suffit dans ce cas de sortir un message d'erreur pour l'utilisateur avant que le programme ne continue son cours normal. Il faut pour ce faire utiliser l'instruction:

RESUME (numéro de ligne)

qui indique à partir de quelle ligne le programme doit se continuer après une erreur. Prenons un exemple concret. Entrez ce programme dans votre ordinateur et lancez-le:

```
10 ON ERROR GOTO 1000
20 FOR I=-1 TO 3
30 ON I GOTO 200,300
40 NEXT I
100 END
200 CLS
210 PRINT "Ligne 210"
220 FOR A=1 TO 2000:NEXT
230 GOTO 40
300 CLS
310 PRINT "Ligne 310"
320 FOR A=1 TO 2000:NEXT
```

```
330 GOTO 40
1000 REM Traitement des erreurs
1010 CLS
1020 PRINT "Erreur";ERR;"en ligne";ERL
1030 FOR A=1 TO 2000:NEXT
1040 RESUME 40
```

Il y a dans ce programme une erreur à la ligne 30. La variable-compteur I a en effet d'abord la valeur -1 alors que l'instruction ON ne marche pas avec les valeurs négatives. C'est pourquoi ce programme sautera, lors du premier parcours de la boucle des lignes 20 à 40, à la routine de traitement des erreurs que nous avons placée en ligne 1000. Cette routine sortira alors le message d'erreur correspondant. L'instruction RESUME conduit alors le programme à continuer l'exécution du programme à partir de la ligne 40. Les lignes 200 puis 300 seront alors exécutées puis le programme sautera normalement à la ligne 100. Comme vous le voyez par cet exemple, l'utilisation des deux instructions de traitement des erreurs est quand même très simple. Dans vos programmes, vous pouvez cependant décrire l'erreur qui s'est produite de manière encore plus claire en faisant sortir un message d'erreur en fonction de la valeur de ERR.

Vous pouvez également mettre fin à un programme en faisant sortir les messages d'erreur "originaux" du CPC avec l'instruction:

```
ERROR X
```

qui affiche le message d'erreur correspondant à la valeur de X puis interrompt le programme.

Voici à nouveau des exercices suivis de leurs solutions. Nous vous invitons à toujours suivre les cinq étapes de programmation que nous avons décrites au début de cet ouvrage. Bonne chance!

EXERCICES

1. Ecrivez un programme qui additionne les nombres de la "série harmonique" (1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n) jusqu'à obtenir une valeur entrée par l'utilisateur du programme. Le nombre d'additions effectuées sera affiché toutes les 50 additions. Il faudra indiquer en fin de programme le nombre des nombres qui ont été additionnés.

2. Ecrivez un programme qui calcule les points zéro réels d'une équation du second degré de forme: AX^2 + BX + C = 0. La formule est:
   x1 = (-B + SQR(B^2-4AC)) / 2A
   x2 = (-B - SQR(B^2-4AC)) / 2A
   Notez qu'il n'y a pas de solutions réelles pour B^2-4AC < 0. Il faut en tenir compte dans votre programme.

3.
```
10 REM EXERCICE TEST AVEC ON ... GOTO
20 INPUT"Entrez un nombre.";Z
30 ON Z GOTO 100,150,400:CLS
40 PRINT"Valeur non-valable."
50 END
100 PRINT"LIGNE 100"
110 END
100 PRINT"LIGNE 100"
.
.
.
```

Que se passera-t-il dans ce programme si vous entrez 4 ? Essayez bien sûr de résoudre ce problème sans entrer le programme dans l'ordinateur.

SOLUTIONS

1.
```
10 REM SERIE HARMONIQUE
20 CLS
30 PRINT "Jusqu'à quelle somme"
40 PRINT:PRINT"faut-il additionner ?"
50 PRINT
60 INPUT S
70 Z=1
80 SH=SH + 1/Z
90 Z=Z+1
100 IF Z = 50 * INT(Z/50) THEN PRINT Z;"ADDITIONS"
110 IF SH < S THEN 80
120 PRINT"Après" Z "termes la somme est de" SH
```

Les lignes 20 à 60 vident l'écran puis demandent l'entrée de la somme à obtenir. La ligne 70 met le compteur à 1 et la ligne 80 additionne les différents termes de l'addition. La ligne 90 augmente alors le compteur de 1. La ligne 100 teste alors si le compteur a atteint un multiple de 50. Si c'est le cas, l'instruction suivant le THEN est exécutée, c'est-à-dire que le nombre d'additions est sorti, comme le demandait l'exercice. La ligne 110 teste si la valeur à obtenir est déjà atteinte. Lorsque la somme voulue a été atteinte, la ligne 120 indique combien de termes ont été nécessaires ainsi que le résultat même de cette addition.

2.
```
10 REM EQUATION DU SECOND DEGRE
20 CLS:PRINT
30 PRINT"Entrée des coefficients A,B,C"
40 PRINT
50 INPUT A,B,C
60 IF A=0 THEN 20:REM A DOIT ETRE <> 0
70 D=B*B-4*A*C
```

```
80 IF D < 0 THEN 140
90 X1 =(-B+SQR(D))/(2*A)
100 X2 =(-B-SQR(D))/(2*A)
110 PRINT"Solution pour X1 =";X1
120 PRINT"Solution pour X2 =";X2
130 GOTO 150
140 PRINT"Pas de points zéro réels !!"
150 END
```

La traduction du problème en programme ne devrait pas vous avoir causé de difficulté. Il faut noter que si A égale 0, il devra être divisé par 2*A suivant la formule. Il y aurait alors une division par 0, ce qui n'est pas autorisé. C'est pourquoi nous avons exclu ce cas au début du programme (ligne 60).

3. L'écran se vide puis le message "Valeur non-valable." est sorti sur l'écran. Il fallait penser que l'instruction placée à la suite du GOTO sera elle aussi exécutée.

3.4 COMMANDE DU DEROULEMENT D'UN PROGRAMME AVEC INKEY$

Nous avons jusqu'ici toujours utilisé l'instruction INPUT pour entrer des données. L'avantage de l'instruction INPUT est qu'elle est facile à employer dans un programme et qu'elle peut être accompagnée d'un texte explicatif. Son inconvénient est qu'on peut détruire l'image du menu en faisant une entrée erronnée ou qu'on peut entrer par erreur une lettre au lieu d'un chiffre. Dans ce cas, le message d'erreur:

```
?Redo from start
```

apparaît. La plupart du temps vous avez alors fait disparaître l'image du menu à l'écran.

Il n'est pas non plus possible d'entrer de virgule avec l'instruction INPUT qui considère les virgules uniquement comme des marques de séparation entre diverses entrées. Vous pouvez toutefois utiliser l'instruction LINE INPUT pour remédier à ce dernier inconvénient, mais vous ne pouvez plus alors entrer qu'une variable. Nous ne voulons toutefois pas du tout vous déconseiller l'emploi de INPUT ou LINE INPUT mais simplement en souligner les limites qui peuvent être plus ou moins gênantes suivant le type de programme que vous réalisez. Mais si vous voulez protéger vos programmes contre une utilisation incorrecte, il vous faut utiliser l'instruction INKEY$.

3.4.1 ENTREE DE DONNEES AVEC INKEY$

L'instruction INKEY$ n'est utilisée le plus souvent, comme l'instruction INPUT, qu'en mode programme. Comment travaille-t-elle?

Il faut savoir que le CPC possède un buffer clavier. Il s'agit d'une petite mémoire qui peut stocker jusqu'à 20 caractères. Pratiquement cela signifie que si l'ordinateur est occupé à exécuter une certaine tâche, vous pouvez entrer jusqu'à 20 caractères au clavier que l'ordinateur utilisera dès qu'il aura terminé sa tâche. Entrez par exemple le programme suivant dans l'ordinateur:

```
10 FOR I=0 TO 10000:NEXT I
```

Une fois lancée cette boucle de temporisation d'environ 10 secondes, appuyez successivement sur les touches T, E, S, T. Une fois la temporisation terminée, vous verrez TEST s'afficher sur l'écran.

Ces 4 lettres ont donc été stockées dans le buffer clavier pendant la durée du programme, où l'ordinateur est ensuite allé les rechercher pour les afficher sur l'écran dès qu'il a eu terminé son travail.

L'instruction INKEY$ vous permet de savoir s'il y a des caractères dans le buffer clavier. Si c'est le cas, INKEY$ lit le premier caractère du buffer et l'affecte à la variable placée à la gauche d'INKEY$. Si le buffer est vide, c'est la valeur 0 qui est affectée à cette variable. L'instruction INKEY$ est une instruction instantanée qui indique uniquement si à l'instant précis où elle est utilisée un caractère se trouve dans le clavier. Elle n'attend pas que vous ayez appuyé sur une touche. Cette instruction ne peut donc avoir d'intérêt pratique que dans une boucle:

```
10 A$=INKEY$:IF A$="" THEN 10
20 PRINT A$
```

Entrez cet exemple et lancez-le. Vous voyez que, contrairement à l'instruction INPUT, INKEY$ ne provoque pas l'apparition du curseur. Notre petit programme attend jusqu'à ce que vous appuyiez sur une touche. La ligne 10 lit essaye en effet de lire un caractère dans le buffer clavier. L'instruction IF ... THEN teste si A$ est une chaîne vide, c'est-à-dire une chaîne de caractères qui ne contient rien. Faites bien attention à ce qu'il n'y ait vraiment rien entre les deux guillemets, même pas un espace ! Si le buffer est vide, le programme saute à nouveau à la même ligne. Si vous appuyez sur une touche, le caractère correspondant sera affiché sur l'écran. Pour bien voir avec quelle rapidité travaille l'instruction INKEY$, modifions notre programme quelque peu :

```
10 CLS
20 A$=INKEY$:Z=Z+1:PRINT CHR$(30)Z:IF A$="" THEN 20
30 PRINT A$
```

Nous avons ajouté un compteur Z en ligne 20 pour compter le nombre de parcours de la boucle jusqu'à ce que vous appuyiez sur une touche. PRINT CHR$(30) place le curseur chaque fois dans l'angle supérieur gauche de l'écran.

Nous pouvons maintenant sélectionner certaines touches, c'est-à-dire n'autoriser que certaines touches bien précises dans nos programmes. La touche ESC constitue toutefois une exception, comme nous le montre cet exemple :

```
10 REM SELECTION DES TOUCHES AVEC INKEY$
20 CLS
30 A$=INKEY$:IF A$="" THEN 30
40 IF A$=CHR$(97) THEN PRINT"Touche a":GOTO 30
50 IF A$=CHR$(98) THEN PRINT"Touche b":GOTO 30
60 IF A$=CHR$(102) THEN END
70 PRINT"Seules les touches a, b ou"
80 PRINT"f pour fin sont permises."
90 GOTO 30
```

Si vous entrez ce programme dans l'ordinateur puis que vous le lancez, vous constaterez que vous ne pouvez influer sur le programme qu'avec les touches a,b ou f ou bien également avec la touche ESC qu'il n'est pas possible d'interroger spécialement.

Voici maintenant un autre programme d'exemple pour sortir le code ASCII d'un caractère ainsi que le caractère lui-même. Comme certains caractères ou touches ont des fonctions de commande (changement de couleur, etc...), nous vous conseillons, après avoir expérimenté ce programme, de remettre l'ordinateur dans son état initial après allumage (CTRL, SHIFT, ESC).

```
10 REM SORTIE DES VALEURS ASCII
20 CLS
30 A$=INKEY$:IF A$="" THEN 30
```

```
40 PRINT TAB(6) CHR$(ASC(A$))TAB(12)ASC(A$)
50 GOTO 30
```

La ligne 40 mérite un commentaire : le caractère est édité avec CHR$ en 6ème position de la ligne. La valeur paramètre de CHR$ est fournie par ASC(A$). Vous ne pouvez voir un caractère que s'il s'agit d'un caractère représentable en mode normal. La seconde partie de la ligne sort la valeur ASCII. La ligne 50 saute alors en ligne 30.

Nous allons maintenant, avec l'exemple suivant, lire 4 caractères avec INKEY$ et les affecter à la variable alphanumérique B$:

```
10 REM LIRE 4 CARACTERES AVEC INKEY$
20 FOR I=1 TO 4
30 A$=INKEY$:IF A$="" THEN 30
40 B$=B$+A$
50 NEXT I
60 PRINT B$
70 END
```

Une boucle FOR ... NEXT permet ici de lire exactement 4 caractères. La ligne 30 affecte le caractère correspondant à la touche sur laquelle vous appuyez à A$. Si vous n'appuyez sur aucune touche, elle saute à la ligne 30 à nouveau. La ligne 40 concatène les caractères ainsi lus et constitue ainsi la chaîne de caractères B$. La ligne 50 ferme la boucle FOR ... NEXT. Une fois que la boucle a été parcourue quatre fois, la variable B$ est sortie.

Nous avons donc illustré quelques exemples d'applications de l'instruction INKEY$. Sa particularité la plus importante est qu'elle permet la sélection des touches, ce qui vous permet d'empêcher totalement que l'image de votre menu ne soit détruite. Nous allons maintenant remplacer dans notre programme de cours de calcul les instructions INPUT par des instructions INKEY$.

La page suivante vous montre comment nous avons remplacé les deux premières instructions INPUT par des instructions INKEY$. Etudiez bien comment les tests IF ... THEN empêche que l'utilisateur n'emploie d'autres touches que celles autorisées. Notez au passage que nous avons été bien inspiré de numéroter notre programme de 10 en 10 car cela nous a permis d'intercaler facilement dans notre programme les lignes supplémentaires nécessaires pour l'instruction INKEY$.

```
.
.
.
120 PRINT
130 PRINT TAB(12)"4 pour la multiplication"
133 PRINT
135 PRINT TAB(12)"5 pour FIN"
140 PRINT
150 PRINT TAB(12)"Quel nombre";Z
155 A$=INKEY$:IF A$="" THEN 155
160 IF VAL(A$)<1 OR VAL(A$)>5 THEN 155
170 ON VAL(A$) GOTO 200,600,1000,1300,1600
200 REM **********
210 REM  ADDITION
220 REM **********
230 CLS
240 PRINT TAB(10)"Entrez le plus grand nombre"
250 PRINT
260 PRINT TAB(10)"pour l'addition."
270 PRINT
290 PRINT TAB(10)"Plus grand nombre"
291 FOR I=1 TO 3
293 A$=INKEY$:IF A$="" THEN 293
295 IF ASC(A$)<48 OR ASC(A$)>57 THEN 293
297 B$=B$+A$:NEXT I
298 GR=VAL(B$)
299 REM
300 REM GENERATION DE NOMBRES ALEATOIRES
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
.
.
.
```

Comme vous le voyez le premier changement concerne les lignes 150-170. L'instruction INPUT avec un message a été scindée en deux: l'instruction PRINT sort maintenant le message et l'instruction INKEY$ en ligne 155 teste si un caractère a déjà été entré, suivant la méthode qui nous est maintenant familière. La ligne 160 convertit le caractère entré en un chiffre avec VAL(A$) et teste s'il est bien à l'intérieur des limites autorisées. VAL(A$) a été utilisé également à la ligne suivante puisque l'instruction ON ... GOTO ne fonctionne qu'avec une variable numérique.

Si vous lancez maintenant cette nouvelle version du programme, vous verrez que vous ne pouvez entrer que les nombres de 1 à 5 pour choisir un point du menu. Toutes les autres touches sont ignorées.

Notre routine INKEY$ a encore un petit défaut: nous ne voyons pas OU nous écrivons sur l'écran ni même ce que nous écrivons. Il nous faut donc afficher nous-même les données entrées, alors que l'instruction INPUT le faisait automatiquement. Mais pour le moment, nous en savons assez pour employer l'instruction INKEY$ dans nos programmes.

3.4.2 INTERROGATION DU CLAVIER AVEC INKEY

Nous avons vu comment nous pouvons contrôler les entrées dans un programme en ne prenant en compte que certaines touches. Nous avons utilisé les codes ASCII (CHR$) pour interroger le clavier.
Le CPC possède cependant également un code propre pour le clavier (voir en annexe) qui est différent du code ASCII. L'instruction INKEY qui vous permet de savoir le code clavier des touches enfoncées présente l'avantage par rapport à INKEY$ de renseigner sur toutes les combinaisons de touches, y compris celles avec SHIFT et CTRL. La syntaxe de cette instruction est:

INKEY(X)

où X représente le code d'une touche, par exemple 52 pour la touche G (voir annexe). Voici un petit programme que vous ne pourrez arrêter qu'en appuyant sur la touche G:

```
10 REM Interrogation du clavier avec INKEY
20 CLS:IF INKEY(52)=0 THEN 30 ELSE 20
30 PRINT"G"
40 END
```

Ce programme une fois lancé, vous ne pouvez l'arrêter qu'en appuyant sur la touche G. Un G s'affichera alors à l'écran. L'ordinateur affichera ensuite Ready et vous verrez alors apparaître un autre G. D'où sort-il ? En fait, l'instruction INKEY n'interroge pas le buffer clavier, de sorte que les caractères que vous tapez sont de toute façon placés dans le buffer clavier et qu'ils seront sortis après la fin du programme. C'est là une différence importante par rapport à l'instruction INKEY$.

Dans notre exemple, nous avons interrogé directement la valeur d'INKEY mais vous pouvez également affecter cette valeur à une variable et tester ensuite la valeur de la variable:

```
10 CLS
20 A=INKEY(52):IF A=0 THEN 30 ELSE 20
30 PRINT"G"
40 END
```

Comme vous le voyez dans ces deux exemples, si une touche est eonfoncée, la valeur INKEY correspondante recevra la valeur 0. Si la touche n'est pas enfoncée, cette valeur sera -1. Si une touche est enfoncée en même temps que la touche SHIFT, la valeur INKEY correspondant à cette touche sera 32, 128 si c'est la touche CTRL qui est enfoncée et 160 si CTRL et SHIFT sont enfoncées simultanément:

COMBINAISON DE TOUCHES	VALEUR CORRESPONDANTE
TOUCHE PAS ENFONCEE	-1
TOUCHE ENFONCEE	0
TOUCHE ENFONCEE + SHIFT	32
TOUCHE ENFONCEE + CTRL	128
TOUCHE ENFONCEE + CTRL + SHIFT	160

Avant d'illustrer par un exemple l'emploi d'INKEY, voici une autre instruction que nous allons utiliser dans notre prochain exemple:

LOCATE X,Y

qui positionne le curseur dans la colonne X et la ligne Y.

```
10 REM TEST DE REFLEXES
20 CLS
30 FOR I=0 TO INT(10000*RND(1))+1:NEXT
40 LOCATE 21,13:PRINT CHR$(224)
50 Depart=INT(TIME/3)
60 A$=INKEY$:Temps=((TIME/3)-Depart):IF A$="" THEN 60
70 LOCATE 1,24
80 PRINT USING"Il vous a fallu ###.## secondes.";Temps/100
90 FOR I=0 TO 4000:NEXT
100 CLS
110 LOCATE 5,5
120 PRINT "Encore une fois (o/n) ?"
130 A$=INKEY$:IF A$="o" THEN 20
140 IF A$="n" THEN 150 ELSE 130
150 END
```

La ligne 30 génère une boucle de temporisation aléatoire. La ligne 40 positionne le curseur en 21ème colonne et 13ème ligne. Le visage (CHR$(224) apparaîtra donc assez exactement au milieu de l'écran après la fin de la temporisation.

La ligne 50 place alors la valeur actuelle de la variable système TIME dans la variable Depart. La ligne 60 interroge alors avec INKEY$ si une touche a déjà été enfoncée. La variable TIME est alors divisée par 3. On obtient une valeur en centièmes de secondes. On soustrait de la valeur ainsi obtenue la valeur de Depart qui contenait le temps écoulé depuis l'allumage de l'ordinateur. La nouvelle valeur calculée est alors affectée à la variable Temps. On teste alors d'après le contenu de A$ si une touche a été enfoncée. La ligne 70 positionne alors le curseur à nouveau et la ligne 80 sort le temps avec un message d'explication. Une autre boucle de temporisation figure en ligne 90 pour permettre la lecture du message. Les autres lignes du programme ne devraient pas vous poser de problème particulier.

Rassurez-vous, vous aurez l'occasion de voir dans nos prochains programmes d'exemple suffisamment d'autres applications des instructions INKEY et INKEY$ pour que vous puissiez en assimiler parfaitement le fonctionnement.

Voici maintenant d'autres instructions très puissantes de votre ordinateur pour la gestion du clavier:

KEY

vous permet d'affecter une fonction à une ou plusieurs touches. Il vous faut pour cela indiquer tout d'abord le code clavier de la touche, puis le texte qui devra être sorti chaque fois que cette touche sera enfoncée. Vous pouvez par exemple affecter à la touche 1 du bloc numérique la fonction de vidage de l'écran:

```
KEY 1,"CLS"+CHR$(13)
```

Le code CHR$(13) (=ENTER) que nous ajoutons à "CLS" fait que chaque fois que vous appuyez sur la touche 1, non seulement CLS s'affiche à l'écran, mais en outre, l'ordinateur effectue un retour de chariot, comme si vous aviez appuyé vous-même sur la touche ENTER, de sorte que l'instruction CLS est immédiatement exécutée.

KEY DEF A,B,X,Y,Z

Cette instruction vous permet d'affecter un autre caractère à une touche. A est le code clavier de la touche, B est le code pour la fonction de répétition (0=non / 1=oui), X est le code ASCII du caractère à affecter à la touche enfoncée seule, Y celui pour la touche avec SHIFT et Z pour la touche avec CTRL:

KEY DEF 52,1,174,174,174

affectera par exemple le point d'interrogation renversé à la touche G.

SYMBOL AFTER et SYMBOL vous permettent de modifier la forme des caractères elle-même:

SYMBOL AFTER X

vous permet de déterminer le code ASCII du caractère à partir duquel une modification de la forme des caractères doit être possible. La valeur standard est 240. Si vous entrez donc:

SYMBOL AFTER 32

vous pourrez redéfinir la forme de tous les caractères dont le code ASCII est supérieur ou égal à 32. Si vous voulez par exemple commencer à créer un jeu de caractères danois, voici déjà les

valeurs créant le A danois avec un cercle:

SYMBOL AFTER 65

SYMBOL 65,24,36,24,60,102,126,102,102

Vous n'avez plus maintenant qu'à appuyer sur la touche A et vous verrez tout de suite la différence.

Nous allons maintenant examiner au chapitre suivant quelques instructions ou fonctions qui sont plus rarement utilisées dans des programmes. Mais il est important que vous connaissiez toutes les possibilités de votre ordinateur au cas où vous en auriez un jour besoin.

3.5 CALL, DI, FRE, POS, UNT, WAIT ETC...

ABS (X)

fournit la valeur absolue d'un nombre X.

PRINT ABS(-464)

donnera donc: 464

CALL (X)

vous permet d'appeler un programme en langage-machine commençant à l'adresse X. Vous pouvez ainsi appeler vos propres programmes en langage-machine ou des routines du système d'exploitation. Vous n'aurez certainement pas à vous servir au début de cette instruction qui n'a d'intérêt que si vous maîtrisez le langage-machine ou si vous connaissez bien le système d'exploitation du CPC.

DI

(Disable Interrupt) interdit toutes les interruptions qui sont provoquées par exemple par les instructions EVERY ou AFTER mais pas celles provoquées par la touche ESC. Les instructions EVERY et AFTER seront traitées dans un chapitre ultérieur.

EI

(Enable Interrupt) met fin à l'effet de DI et autorise donc toutes les interruptions.

FRE(X)

indique le nombre d'octets disponibles en mémoire BASIC. En mode direct, vous pouvez entrer :

PRINT FRE(X)

qui donnera le nombre d'octets encore disponibles pour le BASIC (43533 à l'allumage de votre ordinateur). X peut être ici remplacé par n'importe quelle valeur ou variable sans que cela change le résultat.

INP

fournit la valeur se trouvant dans le canal d'entrée-sortie donné comme paramètre. Vous pouvez entrer par exemple :

PRINT INP(&FF00)

JOY

fournit la valeur correspondant à la position du joystick (manche à balai). Le programme suivant vous donne les valeurs correspondant aux différentes positions du joystick :

```
10 CLS
20 PRINT CHR$(30);JOY(0)
30 GOTO 20
```

OUT

envoie une valeur entre 0 et 255 à l'adresse fournie comme paramètre. Par exemple :

```
OUT &FF0A,205
```

PEEK

fournit le contenu d'une adresse quelconque de la mémoire du CPC. Par exemple :

```
PRINT PEEK(368)
```

POKE

écrit une valeur donnée dans une adresse quelconque de la mémoire :

```
POKE 366,10
```

écrira un 10 à l'adresse 366.

POS

fournit la position actuelle du curseur sur la ligne de l'écran sur laquelle il se trouve. Entrez par exemple en mode direct :

PRINT "TEST" POS(#0);"TEST A" POS(#0)

et appuyez sur la touche ENTER. Vous obtenez :

TEST 5 TEST A 14

Le 5 correspond à la colonne où se trouvait le curseur après exécution de la première instruction PRINT, le 14 à celle où il était après la seconde instruction PRINT. Avec POS, vous devez toujours indiquer un numéro de périphérique de sortie (#X). Dans notre exemple il s'agissait du périphérique 0, l'écran.

VPOS

fournit la position verticale actuelle du curseur sur la colonne de l'écran sur laquelle il se trouve. Avec VPOS, vous devez également indiquer toujours un numéro de périphérique de sortie (#X).

SPEED KEY

vous permet d'influer sur le délai et la vitesse de répétition des touches. L'unité utilisée correspond à deux centièmes de secondes. Si vous entrez par exemple :

SPEED KEY 10,5

toute touche enfoncée sera répétée après un délai de 2 dixièmes de seconde tous les dixièmes de seconde.

SPEED WRITE

vous permet de fixer à quelle vitesse doit s'effectuer un stockage de données sur cassette. La valeur standard est de 1000 baud (bits par seconde) et correspond à SPEED WRITE 0. SPEED WRITE 1 fait passer la vitesse à 2000 baud.

UNT

convertit les valeurs comprises entre 0 et 65535 en valeurs comprises entre -32768 et +32767. Par exemple :

PRINT UNT(50000)

donnera: -15536

WAIT

attend qu'une adresse d'entrée-sortie ait pris une valeur entre 0 et 255.

WIDTH

Cette instruction limite le nombre maximum de caractères pouvant être sortis sur une ligne lors d'une sortie sur imprimante. Par exemple :

WIDTH 90

3.6 READ, DATA ET RESTORE

Nous n'avons jusqu'ici appris à entrer des données qu'à partir du clavier avec les instructions INPUT et INKEY$. Mais si notre programme doit traiter un grand nombre de données, qu'il s'agisse de variables numériques ou de chaînes de caractères, il est bien sûr très pénible de devoir entrer ces données chaque fois qu'on relance le programme. Les instructions READ et DATA permettent de résoudre ce problème.

L'instruction DATA se compose de listes de données numériques ou alphanumériques (chaînes de caractères) séparées entre elles par des virgules.

READ vous permet de lire les données dans l'ordre où elles figurent dans les instructions DATA. Il faut cependant faire attention à ce que le type de variable suivant l'instruction READ corresponde bien au type de donnée à lire. Vous ne pouvez pas faire lire une chaîne de caractères placée en DATA par une variable numérique.

Les lignes de DATA peuvent figurer en n'importe quel endroit du programme, au début, à la fin ou au milieu. Si le programme rencontre une instruction READ, il va rechercher automatiquement la première donnée placée dans une instruction DATA qu'il n'a pas encore lue. Il affecte alors cette donnée à la variable suivant l'instruction READ. Par exemple :

```
10 READ X
20 PRINT X
30 DATA 50
40 END
```

Vous obtenez 50. La ligne 10 affecte en effet à la variable X la valeur 50 lue par l'instruction READ.

La ligne 20 sort alors la valeur de X et la ligne 30 n'a plus aucune influence sur le déroulement ultérieur du programme. Modifiez par exemple votre programme ainsi :

```
10 READ X,Y,Z
20 PRINT X,Y,Z
30 DATA 10,20,30
40 END
```

Après avoir lancé le programme, vous obtenez à l'écran :

```
10          20          30
```

READ a donc lu d'abord la première donnée figurant en DATA (10) qu'il a affectée à la variable X. READ a ensuite lu la seconde donnée figurant en DATA (20) qu'il a affectée à la variable Y. READ a lu enfin la troisième donnée figurant en DATA (30) qu'il a affectée à la variable Z. L'ordinateur lit donc chaque fois la donnée suivante grâce à ce qu'on appelle un pointeur, qui est dirigé sur la prochaine donnée à lire après chaque instruction READ. Au début du programme, ce pointeur est donc dirigé sur le premier élément de la première ligne de DATA. En voici une illustration (le pointeur sera représenté par une flèche) :

```
30 DATA 10,20,30
      ^
```

Après l'exécution de la première instruction READ du programme, le pointeur est augmenté :

```
30 DATA 10,20,30
        ^
```

Après l'exécution de la seconde instruction READ du programme, le pointeur est à nouveau augmenté. Mais lorsque l'ordinateur arrive à la fin d'une liste de données, le pointeur n'est pas automatiquement dirigé à nouveau sur le premier élément mais il reste au contraire dirigé sur ce qui suit le dernier élément. Si on tente alors à nouveau de lire une donnée de la liste avec READ, l'ordinateur sort le message :

DATA exhausted in (numéro de ligne)

(données épuisées). Si vous voulez utiliser plusieurs fois une même liste, il faut utiliser l'instruction :

RESTORE

qui dirige à nouveau le pointeur sur le premier élément de la liste. Mais entrez d'abord le programme suivant pour voir ce qui se produit lorsque le programme essaye de lire plus de données que la liste n'en comporte :

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
40 READ D,E,F
50 PRINT D,E,F
60 END
```

Après que les valeurs 10, 20 et 30 aient été sorties, le message d'erreur :

DATA exhausted in 40

apparaît La ligne 40 a en effet essayé de lire un quatrième élément de DATA alors qu'il n'y en avait que 3.

Vous pouvez remédier à cette erreur soit en ajoutant trois données supplémentaires en DATA, soit en utilisant RESTORE pour réinitialiser le pointeur de données :

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
35 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 END
```

Dans cette nouvelle version (vous n'avez à entrer que la ligne 35) le programme fonctionne sans erreur et sort :

```
10        20        30
10        20        30
```

Le pointeur de données a en effet été réinitialisé et les variables D, E et F se sont également vu affecter les valeurs 10, 20 et 30.
Notez qu'une seule instruction permet de lire plusieurs données et de les affecter à plusieurs variables différentes, comme nous l'avons fait avec READ A,B,C et READ D,E,F. Vous pouvez également affecter les différentes valeurs à lire successivement à la même variable :

```
10 FOR I=1 TO 3
20 READ X
30 PRINT X
40 NEXT I
50 DATA 10,20,30
60 END
```

Nous avons ici placé les instructions READ X et PRINT X dans une boucle qui sera parcourue trois fois.

Lors de chaque parcours, c'est la prochaine valeur figurant en DATA qui sera affectée à l'instruction X puis sortie avec PRINT X.

Comme nous l'avons déjà indiqué, vous pouvez également placer des chaînes de caractères en DATA. Celles-ci n'ont pas en principe à être mises entre guillemets. Mais il y a comme toujours des exceptions à respecter pour éviter des déboires :

La virgule, le double-point et le point-virgule doivent être entre guillemets. Mais pensez surtout à ne pas lire avec une variable numérique une chaîne de caractères. Sinon le programme sera en effet interrompu par le message d'erreur :

Syntax error in (numéro de ligne)

Si vous avez créé une longue liste de données mixtes, une telle erreur peut se produire facilement :

```
10 FOR I=1 TO 3
20 READ A,B,C$
30 PRINT A,B,C$
40 NEXT I
50 DATA 10,20,TEST 1,30,40,TEST 2,50,TEST 3,OK
60 END
```

Jusqu'au deuxième parcours de la boucle FOR ... NEXT, notre programme tournera sans problème car jusque là les types de variables correspondent aux données effectivement placées en mémoire : l'instruction READ prévoit que deux variables numériques doivent être lues puis une variable alphanumérique et ainsi de suite. Les données placées en ligne 50 correspondent à cet ordre jusqu'à la septième donnée, 50.

Mais ensuite, le programme va essayer d'affecter la chaîne de caractères TEST 3 à la variable numérique B, ce qui n'est pas possible, comme vous le savez et qui provoque donc une interruption du programme lors du troisième parcours de la boucle avec le message d'erreur :

Syntax error in 50

Nous avons appris beaucoup de choses sur la transmission de données à l'ordinateur ou au programme. Nous pouvons entrer des données directement avec INPUT ou INKEY$ ou les placer dans des lignes de DATA qui seront d'ailleurs sauvegardées avec le programme si vous le sauvegardez sur cassette ou sur disquette! Par contre, pour sauvegarder des données entrées avec INPUT ou INKEY$, il faut les écrire sur un fichier séparé sur cassette ou disquette.

Il nous faut signaler une application très courante des instructions READ et DATA que vous n'utiliserez toutefois peut-être pas dans l'immédiat et qui consiste à générer des programmes en langage-machine à partir du BASIC. Il s'agit le plus souvent de nombreuses lignes de DATA qui sont lues par une boucle FOR ... NEXT et placées par l'instruction POKE dans les adresses correspondantes. Après quoi le programme en langage-machine se trouve en mémoire et peut être appelé avec l'instruction CALL.

En tout cas les instructions READ, DATA et RESTORE sont très importantes et nous vous invitons à relire attentivement ce chapitre si vous n'êtes pas sûr d'avoir tout assimilé parfaitement. Nous allons en effet aborder maintenant les tableaux et vous verrez que nous aurons de nombreuses occasions de recourir aux instructions READ et DATA.

4. APPLICATIONS BASIC PLUS COMPLEXES

4.1 LES TABLEAUX

La programmation des tableaux est certainement la technique la plus difficile qu'il soit nécéssaire d'assimiler pour un débutant en BASIC. C'est pourquoi il est nécessaire d'aborder cette technique avec un esprit de méthode. Comme toujours, nous vous guiderons par des exemples très simples au départ puis de plus en plus complexes.

4.1.1 TABLEAUX A UNE DIMENSION

Supposez que vous vouliez écrire un programme calculant votre revenu mensuel moyen sur une période de 12 mois. Nous utiliserons tout d'abord une boucle pour entrer les 12 revenus mensuels :

```
10 REM revenu mensuel moyen
20 REM calcule sur 12 mois
30 FOR I=1 TO 12
40 INPUT"Revenu mensuel";M
50 S=S+M
60 NEXT I
70 DU=S/12
80 DU=INT(DU*100)/100
90 PRINT"Revenu moyen";
100 PRINT DU;"F"
110 END
```

Après avoir lancé ce programme, entrez 12 valeurs. Vous obtenez le revenu mensuel moyen arrondi à deux décimales. Une boucle FOR ... NEXT entre les différentes valeurs des revenus mensuels et la ligne 50 calcule au fur et à mesure la somme de ces revenus. La ligne 70 calcule le revenu mensuel moyen et la ligne 80 l'arrondit à deux décimales.

Nous avons donc bien calculé le revenu mensuel moyen, mais nous ne pourrions plus obtenir du programme le détail des différents revenus mensuels, par exemple le revenu du mois de mai. Pour conserver les valeurs des différents revenus mensuels, il faudrait que nous n'utilisions pas une seule et même variable pour entrer ces revenus. Il suffirait donc d'utiliser 12 variables, une pour chaque mois :

```
10 REM revenu mensuel moyen
20 REM calcule sur 12 mois
30 INPUT"Revenu mensuel 1";M1
40 INPUT"Revenu mensuel 2";M2
50 INPUT"Revenu mensuel 3";M3
60 INPUT"Revenu mensuel 4";M4
70 INPUT"Revenu mensuel 5";M5
80 INPUT"Revenu mensuel 6";M6
90 INPUT"Revenu mensuel 7";M7
100 INPUT"Revenu mensuel 8";M8
110 INPUT"Revenu mensuel 9";M9
120 INPUT"Revenu mensuel 10";M10
130 INPUT"Revenu mensuel 11";M11
140 INPUT"Revenu mensuel 12";M12
150 S=M1+M2+M3+M4+M5+M6+M7+M8+M9+M10+M11+M12
160 DU=S/12
170 DU=INT(DU*100)/100
180 PRINT"Revenu moyen";
190 PRINT DU;"F"
200 END
```

Si vous développez ce programme, vous pourrez réutiliser les valeurs pour les différents mois à tout moment, grâce aux 12 variables M1 à M12. Pour connaître la valeur du mois de mai, il vous suffira d'utiliser la variable M5.

Nous avons donc maintenant en permanence les 12 revenus mensuels à notre disposition, mais cela nous a tout de même coûté 9 lignes de programmation supplémentaires. D'autre part le maniement de 12 variables n'est pas très pratique: si vous voulez sortir plus tard les 12 valeurs correspondant aux revenus mensuels, vous devrez utiliser une instruction telle que celle-ci :

```
100 PRINT M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12
.
.
.
```

Evidemment le BASIC offre une solution plus élégante et plus pratique, les variables indicées ou tableaux. Vous pouvez ainsi créer des variables telles que celle-ci :

A(I)

où I est l'indice de la variable. Vous pouvez donc, pour rester dans notre exemple, faire entrer les 12 revenus avec une boucle qui fasse varier la valeur de I de 1 à 12 et vous pourrez ensuite demander la valeur de n'importe quel mois en utilisant l'indice de ce mois. La valeur du mois de mai par exemple nous sera donc fournie par A(5).

Nous avons comparé, au début de cet ouvrage, les variables à des tiroirs dans lesquels on peut placer, selon le type de variable, des valeurs numériques ou alphanumériques. On pourrait de même comparer un tableau à un placard à plusieurs tiroirs placés les uns au-dessus des autres. Chaque tiroir porterait un numéro qui n'aurait rien à voir avec le contenu de ces tiroirs. Ce numéro est un INDICE. L'indice est placé entre parenthèses à la suite de la variable indicée ou tableau :

A(1)

1 est ici l'indice de la variable. Il ne faut pas confondre A(1) et A1! A1 est une variable ordinaire alors que A(1) est une des variables indicées du tableau A(X). Voici comment on pourrait représenter un tel tableau :

A(1)	2334
A(2)	2333
A(3)	2345.65
A(4)	2344.34
. . . .	. . . .
A(12)	3433.20

Vous voyez que le terme de tableau décrit bien la réalité. Notre tableau a ici 12 tiroirs dont chacun contient une valeur. Si nous voulons connaître la valeur du troisième élément, il nous suffit d'utiliser l'indice 3 :

PRINT A(3)

donnera donc :

2345.65

Mais si nous voulons utiliser de tels tableaux dans nos programmes, il nous faut indiquer tout d'abord à l'ordinateur les dimensions de notre tableau, c'est-à-dire combien d'éléments il devra contenir. Il nous faut pour cela employer l'instruction DIM dont la syntaxe est la suivante :

DIM nom du tableau (nombre d'éléments)

Dans notre exemple, nous devrions utiliser l'instruction :

DIM A(12)

où A est le nom du tableau et 12 le nombre maximum d'éléments ou de champs du tableau. L'instruction DIM est le plus souvent placée en début de programme. En effet, une fois qu'un tableau a été dimensionné, vous ne pouvez le redimensionner pendant tout le déroulement du programme. Sinon vous obtenez le message d'erreur :

Array already dimensionned in (numéro de ligne)

Nous avons, dans l'exemple ci-dessus dimensionné un tableau de variables à virgule flottante, c'est-à-dire de variables pouvant recevoir n'importent quels nombres réels. Voici des exemples de dimensionnement de tableaux de variables entières, alphanumériques (chaînes de caractères) et à nouveau numériques avec virgule flottante :

DIM DE$(15)
DIM GZ%(20)
DIM AB(12)

Vous pouvez dimensionner plusieurs tableaux avec une seule instruction DIM :

DIM A(12),B$(16),S%(20)

Il convient de noter les deux particularités suivantes :

1. Si vous n'avez pas besoin de plus de 11 éléments ou champs dans votre tableau, il n'est pas nécessaire de le dimensionner. Il suffira en effet que vous appeliez un élément quelconque de ce tableau, par exemple A(4) pour que l'ordinateur exécute automatiquement l'instruction, en l'occurence, DIM A(10).

2. L'indice de tout tableau commence à 0. La valeur utilisée pour dimensionner un tableau doit donc être inférieure de 1 au nombre d'éléments ou de champs que devra contenir votre tableau. Par exemple, DIM A(10) crée un tableau de 11 variables indicées, de A(0) à A(10).

Pour le moment nous ignorerons l'élément d'indice 0 du tableau, comme nous l'avons fait dans la figure précédente.

Si le dimensionnement d'un tableau de moins de 12 éléments ou champs est facultatif, il peut toutefois économiser de la place en mémoire si vous êtes sûr que votre tableau aura moins de 11 éléments, 6 par exemple. Dans ce cas vous pouvez utiliser l'instruction DIM X(6) par exemple.

Voyons maintenant ce que deviendrait notre programme de calcul du revenu mensuel moyen avec un tableau :

```
10 REM revenu mensuel moyen
20 REM avec utilisation d'un tableau
30 DIM M(12)
40 FOR I=1 TO 12
```

```
50 INPUT"Revenu mensuel";M(I)
60 S=S+M(I)
70 NEXT I
80 DU=S/12
90 DU=INT(DU*100)/100
100 PRINT"Revenu moyen";
110 PRINT DU;"F"
120 END
```

Pour utiliser un tableau, nous n'avons eu besoin que d'une ligne d'instruction de plus que dans la première version du programme, pour introduire l'instruction DIM. Nous pouvons maintenant appeler à tout moment les variables correspondant aux différents revenus mensuels. Par exemple, si nous voulions sortir à nouveau un récapitulatif de tous les revenus mensuels avant de sortir le revenu mensuel, il suffirait de modifier ainsi la dernière partie du programme :

```
.
.
.
90 DU=INT(DU*100)/100
100 FOR I=1 TO 12
110 PRINT"REVENU MENSUEL"I,M(I)
120 NEXT I
130 PRINT"Revenu moyen";
140 PRINT DU;"F"
150 END
```

Nous avons donc ainsi trouvé avec les tableaux un moyen simple et pratique de garder disponibles pour une éventuelle utilisation ultérieure les revenus des différents mois.

Les tableaux qui s'écrivent ainsi :

A(X)

sont appelés tableaux unidimensionnels parce qu'ils ne possèdent qu'un indice.

L'indice peut être fourni aussi bien sous forme d'une variable ou d'une expression arithmétique que sous forme d'un nombre. Ceci vous permet de faire calculer l'indice en fonction de vos besoins. La même remarque s'applique également au nombre de champs ou éléments défini par l'instruction DIM. Vous pourriez donc parfaitement prévoir, toujours dans notre exemple de revenu mensuel moyen, que l'utilisateur puisse déterminer le nombre de mois dont il veut la moyenne :

```
.
.
.
30 INPUT"Combien de revenus mensuels";Z
40 DIM M(Z)
.
.
.
```

Le nombre de mois détermine ici les dimensions du tableau, ce qui permet d'adapter le programme aux besoins actuels de l'utilisateur et d'utiliser la mémoire disponible d'une façon optimale.

Si vous essayez d'appeler un élément d'un tableau qui sort du cadre fixé par l'instruction de dimensionnement DIM A(X), vous obtenez le message d'erreur:

Subscript out of range

Par exemple, vous ne pouvez pas appeler l'élément A(16) si vous avez dimensionné le tableau A avec l'instruction DIM A(15).

Pour bien assimiler le fonctionnement des tableaux qui constituent une possibilité très importante de la programmation en BASIC, nous allons l'illustrer par quelques exemples avec des tableaux unidimensionnels. Nous utiliserons chaque fois les tableaux X et Y$ avec toujours 6 éléments. Nous continuerons pour le moment d'ignorer l'élément d'indice 0.

4.1.2 EXEMPLES D'APPLICATIONS DES TABLEAUX UNIDIMENSIONNELS

Lorsque vous venez de créer pour la première fois un tableau donné, tous ses éléments sont vides. Mais il se peut que vous ayez besoin d'annuler le contenu d'un tableau au cours du déroulement d'un programme. S'il s'agit d'un tableau numérique, vous pouvez annuler son contenu en affectant un 0 à tous les éléments du tableau :

```
10 REM EXEMPLE d'annulation TABLEAU NUMERIQUE
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=0
50 NEXT I
60 END
```

Nous sommes parti dans cet exemple d'un tableau de 6 éléments (parce que nous oublions pour le moment le champ 0). La boucle FOR ... NEXT a reçu pour valeurs de départ et de fin 1 et 6, de façon à ce que le compteur I prenne successivement la valeur de tous les indices du tableau, de 1 à 6. La ligne 40 annule lors de chaque parcours l'élément d'indice I. Après le dernier parcours de la boucle, tous les éléments du tableau ont été ainsi annulé. Si nous n'avions pas utilisé une boucle, nous aurions dû écrire :

```
.
X(1)=0
X(2)=0
X(3)=0
X(4)=0
X(5)=0
X(6)=0
.
```

S'il s'agit maintenant d'annuler le contenu d'un tableau de variables alphanumériques (chaînes de caractères), vous ne pouvez bien sûr procéder de la même manière. Ce n'est pas le 0 (qui est aussi un caractère) ni même un espace mais une chaîne vide qu'il faut affecter à chaque élément du tableau :

```
10 REM EXEMPLE d'annulation TABLEAU ALPHANUMERIQUE
20 DIM Y$(6)
30 FOR I=1 TO 6
40 Y$(I)=""
50 NEXT I
60 END
```

Rappelons que c'est une chaîne vide qu'il faut affecter à chaque élément du tableau. Il ne doit donc pas y avoir d'espace entre les deux guillemets à la ligne 40.

Si vous utilisez cette méthode d'annulation d'un tableau numérique ou alphanumérique, vous ne supprimez que le contenu de ce tableau, en conservant le dimensionnement du tableau intact. Si par contre vous voulez annuler également le dimensionnement du tableau, par exemple pour le redimensionner en fonction de nouveaux besoins, il vous faut utiliser l'instruction ERASE qui annule complètement un tableau, y compris son dimensionnement.

Venons-en maintenant à des exemples qui vous permettront de vous exercer au maniement des tableaux avec des boucles FOR ... NEXT. Soient 3 tableaux de 6 éléments chacun dont le contenu sera le suivant :

a)	b)	c)
1	10	2
4	8	4
9	6	8
16	4	16
25	2	32
36	0	64

Comment pourrait-on créer ces trois tableaux avec une boucle FOR ... NEXT.

Exemple a)

Si vous réfléchissez un peu, vous constaterez que les contenus des divers éléments correspondent au carré de l'indice. Ce tableau aurait donc pu être créé avec un programme tel que celui-ci :

```
10 DIM X(6)
```

```
20 FOR I=1 TO 6
30 X(I)=I*I
40 NEXT I
50 END
```

Le fonctionnement de cette boucle est encore plus évident si on écrit sur le papier ce que le programme fait lors de chaque parcours de la boucle :

CREATION DU TABLEAU X(6)

I = 1	X(1) = 1*1 = 1	1
I = 2	X(2) = 2*2 = 4	4
I = 3	X(3) = 3*3 = 9	9
I = 4	X(4) = 4*4 = 16	16
I = 5	X(5) = 5*5 = 25	25
I = 6	X(6) = 6*6 = 36	36

I est augmenté de 1 après chaque parcours de la boucle. C'est ainsi que nous pouvons atteindre chaque élément du tableau. En même temps, nous multiplions chaque fois I par lui-même et nous affectons le résultat à la variable d'indice I.
C'est le même principe, l'utilisation du compteur pour calculer les valeurs des différents éléments du tableau qui a été utilisé pour l'exemple b.

Exemple b)

Dans cet exemple, les différents éléments du tableau diminuent de 2 en partant de 10 chaque fois que l'indice augmente de 1. Nous ne pouvons pas utiliser directement l'indice pour obtenir ces valeurs :

FOR I=10 TO 0 STEP -2 FAUX!

car dans ce cas, nous arriverions à un tout autre résultat: chaque élément du tableau recevrait en effet une valeur égale à son indice et nous aurions un tableau comportant les éléments 10 à -2, ce qui n'est de toute fa_on pas possible. Il nous faut donc trouver une formule qui affecte à chaque élément du tableau d'indice 1 à 6 les valeurs 10 à 0 :

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=12 - 2*I
40 NEXT I
50 END
```

Nous avons placé la formule en ligne 30. Vous voyez que nous avons une fois encore utilisé l'indice pour calculer la valeur des différents éléments. Vous pouvez contrôler la justesse de notre formule en complètant le programme avec les lignes suivantes qui sortiront le contenu de tous les éléments du tableau :

```
50 FOR I=1 TO 6
60 PRINT X(I)
70 NEXT I
80 END
```

Si vous avez du mal à comprendre d'emblée le fonctionnement de notre formule, vous pouvez utiliser la méthode que nous avons employée pour l'exemple précédent et écrire sur le papier ce que donnera cette formule pour chaque valeur de l'indice I. Voyons maintenant l'exemple c) où vous devriez également avoir décelé une certaine régularité.

Exemple c)

Si cette suite de nombres vous semble familière, c'est qu'il s'agit en effet des puissances de 2 que nous avons déjà rencontrées dans le chapitre consacré aux systèmes numériques. Il va donc être très facile de trouver une formule de calcul des différentes valeurs des éléments de notre tableau à partir de la valeur du compteur I :

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=2^I
40 NEXT I
50 END
```

Vous pouvez ici aussi contrôler la justesse de la formule en ajoutant les lignes de programme que nous avons utilisées à l'exemple précédent. Vous pouvez également écrire sur le papier le résultat de la formule pour les valeurs successives de I.

Cette technique de calcul des valeurs d'un tableau à partir de la valeur d'un indice est très importante. Assurez-vous donc que vous l'avez parfaitement comprise. Vous n'aurez plus dès lors de difficulté à comprendre la logique de programmes plus complexes qui utilisent cette technique.

Nous n'avons jusqu'ici pris comme exemples que des tableaux numériques.

En ce qui concerne les tableaux alphanumériques, leur contenu ne présente pas en principe ce caractère de régularité qui est parfaitement possible dans les tableaux numériques. Le contenu des différents éléments d'un tableau alphanumérique est souvent entré au moyen du clavier. Un tableau alphanumérique peut également être "chargé" à partir de lignes de DATA et de l'instruction READ.

Les tableaux alphanumériques sont utilisés par exemple pour stocker dans la mémoire de l'ordinateur des noms, des adresses ou des valeurs numériques (sous forme de chaînes de caractères). L'utilisation des tableaux alphanumérique est donc plus variée que celie des tableaux numériques.

Nous allons créer un petit programme d'exemple qui nous permette d'entrer dans l'ordinateur les prénoms de nos amis. Bien sûr ce programme sera sans intérêt pour le moment puisque nous ne savons pas encore comment sauvegarder ces données sur un périphérique de stockage. Mais cet exemple nous sera très utile pour mieux comprendre plus tard les programmes de gestion de fichier.
Comme vous n'avez certainement pas en tête le nombre précis d'amis dont vous voudriez entrer le nom, ce type de programme ne doit pas faire appel à l'utilisateur pour dimensionner le tableau mais lui indiquer simplement quand il dépasse la capacité de ce tableau. Voici comment un tel programme pourrait être construit :

```
10 REM Liste de prenoms
20 DIM Y$(6)
30 Z=Z+1
40 INPUT"Prenom";Y$(Z)
50 PRINT"Autres entrees o/n ?"
60 A$=INKEY$:IF A$="" THEN 60
70 IF A$<>"o" THEN 100
80 IF Z<6 THEN 30
90 PRINT"Liste pleine!"
100 END
```

Nous avons pour plus de clarté choisi de n'affecter que 6 éléments à notre tableau (ligne 20). La ligne 30 contient le compteur qui est augmenté de 1 lors de chaque parcours. Comme nous ne connaissons pas le nombre exact de prénoms que nous voulons entrer, nous ne pouvons pas utiliser une boucle FOR ... NEXT. La ligne 40 demande donc avec INPUT l'entrée d'un prénom et l'affecte à l'élément du tableau d'indice Z. La ligne 50 demande si d'autres entrées doivent être faites. La ligne 70 compare le caractère entré à "o"; si ce caractère est différent de "o", le programme saute à la ligne 100 qui met fin au programme. Si ce caractère est "o", la ligne 80 compare le compteur à 6. Si le compteur a déjà atteint la valeur 6, la ligne 90 sort le message "Liste pleine" et le programme se termine. Si nous n'avions pas prévu ce test sur la valeur du compteur Z, le programme serait interrompu après la septième entrée par le message d'erreur :

Subscript out of range in 40

Si en effet Z atteint la valeur 7, la ligne 40 tente d'appeler l'élément Y$(7) qui n'existe pas d'après l'instruction DIM de la ligne 20.

Ce programme n'est bien sûr pas particulièrement performant. Vous pourriez l'améliorer en demandant en fin de programme s'il faut sortir la liste de tous les prénoms entrés. Vous pouvez utiliser une instruction IF ... THEN pour cette demande et une boucle FOR ... NEXT pour sortir les différents éléments du tableau.

Vous pouvez également augmenter la puissance de votre programme, par exemple avec l'instruction DIM Y$(200), ce qui sera certainement suffisant pour une gestion de données personnelle. Mais si vous voulez pouvoir entrer d'autres données que les prénoms de vos amis, un tableau unidimensionnel ne suffira pas.

Voyons maintenant un exemple de création d'un tableau avec les instructions READ et DATA. Supposons par exemple que nous ayons souvent besoin dans un programme d'utiliser les noms des jours de la semaine. Il est inutile d'entrer les noms de la semaine chaque fois que vous lancerez votre programme puisque vous pouvez les placer dans des lignes de DATA et les faire lire au début du programme avec l'instruction READ :

```
10 REM Jours de la semaine
20 DIM jours$(7)
30 FOR I=1 TO 7
40 READ jours$(I)
50 NEXT I
60 DATA Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi,Dimanche
70 REM Sortie oui/non
80 PRINT"Sortie du tableau o/n ?"
90 A$=INKEY$:IF A$="" THEN 90
100 IF A$<>"o" THEN 90
110 FOR I=1 TO 7
120 PRINT jours$(I)
130 NEXT I
140 END
```

Ce programme ressemble beaucoup au programme permettant d'entrer une liste de prénoms. La différence essentielle est la ligne 40 qui lit les différents éléments du tableau non pas avec une instruction INPUT mais avec une instruction READ. La fin du programme permet à l'utilisateur de faire sortir la totalité du tableau.

Voici maintenant des exercices qui doivent vous permettre de vous assurer que vous avez bien compris le principe et l'utilisation des tableaux à une dimension, avant que nous n'abordions les tableaux à plusieurs dimensions. Bonne chance!

EXERCICES

1) Ecrivez un programme qui entre 6 noms et les place dans un tableau. Le programme doit également sortir le premier nom dans l'ordre alphabétique. Essayez ce programme avec les noms Raoul, Pierre, Henri, Christine, France et Hyacinthe. N'oubliez pas que vous pouvez comparer les chaînes de caractères entre elles. Le programme devra ici sortir "Christine".

2) Ecrivez un programme qui génère 6 nombres aléatoires compris entre 50 et 150 et les place dans un tableau. Le programme devra également sortir le plus grand de ces nombres.

3) Soit le tableau suivant X(6):

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)
0	2	6	12	20	30

Ecrivez un programme avec une formule qui permette de créer ce tableau. Faites sortir le tableau pour pouvoir contrôler la justesse de la formule.

SOLUTIONS

1.
```
10 REM Entree de noms
20 DIM Y$(6)
30 FOR I=1 TO 6
40 INPUT"Nom";Y$(I)
50 NEXT I
60 REM Premier nom dans l'ordre alphabétique
70 Y$(0)=Y$(1)
80 FOR I=2 TO 6
90 IF Y$(0) <= Y$(I) THEN 110
100 Y$(0)=Y$(I)
110 NEXT I
120 PRINT "Le premier nom est ";Y$(0)
130 END
```

La première partie du programme ne présentait pas de difficulté puisque nous avons utilisé la même technique dans plusieurs exemples. Nous avons pu utiliser ici une boucle FOR ... NEXT puisque nous savions qu'il fallait entrer exactement 6 noms. La deuxième partie était plus complexe et vous pouvez être fier de vous si vous y êtes arrivé également. Nous avions déjà parlé du stockage provisoire de valeurs et de données et c'est cette technique que nous avons utilisée ici.

L'important est qu'aucune donnée ne soit perdue, quelle que soit la variable que vous ayez utilisé pour le stockage provisoire des données. Nous avons utilisé l'élément 0 du tableau puisqu'il n'a pas été employé jusqu'ici. La ligne 70 "sauve" donc le contenu de Y$(1) en Y$(0). En ligne 80 commence la boucle FOR ... NEXT avec la valeur de départ 2. Il n'est en effet pas utile de comparer le premier élément avec lui-même. La ligne 90 compare ensuite les divers éléments du tableau, dans l'ordre où ils ont été entrés.

Si le nom qui se trouve en Y$(0) est "déjà" plus petit que celui qui se trouve momentanément en Y$(I), le programme saute à la ligne 110 et la variable-compteur est augmentée de 1.
Si le nom en Y$(0) est par contre plus grand que celui qui se trouve momentanément en Y$(I), la valeur de Y$(I) est affectée à Y$(0). Lorsque la variable-compteur atteint la valeur 6, le nom recherché se trouve en Y$(0). La ligne 110 peut alors le sortir avec un message de commentaire.

2.
```
10 REM Entree de nombres
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=INT(100*RND(1))+50
50 NEXT I
60 REM Chercher le plus grand nombre
70 X(0)=X(1)
80 FOR I=2 TO 6
90 IF X(0) <= X(I) THEN 110
100 X(0)=X(I)
110 NEXT I
120 PRINT "Le plus grand nombre est";X(0)
130 END
```

La structure de ce programme est identique à celle de l'exemple précédent. Si vous aviez résolu le premier exercice, vous devez donc avoir aussi résolu le second. Les seules différences entre les deux programmes tiennent à la différence de type de tableau (ici tableau numérique) et à la génération de nombres aléatoires en ligne 40. Les comparaisons pour rechercher le plus grand nombre suivent le même principe que la recherche du premier nom dans l'ordre alphabétique dans l'exemple précédent.

3.
```
10 Formule de la serie
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=I*I-I
50 NEXT I
60 REM Sortie du tableau
70 FOR I=1 TO 6
80 PRINT X(I)
90 NEXT I
100 END
```

La structure du programme ne pose pas de problème particulier. Remarquons simplement que vous pouvez avoir trouvé une autre formule pour calculer les valeurs de notre tableau. La seule chose qui compte est que votre programme sorte bien les valeurs que nous vous avions données.

4.1.3 LES TABLEAUX A PLUSIEURS DIMENSIONS

Nous n'avons utilisé jusqu'ici que des tableaux à une dimension. Si nous comparons les tableaux à des listes, il est cependant évident que les listes utilisées dans la vie courante comportent le plus souvent plusieurs informations à chaque ligne. Elles se composent donc de LIGNES et de COLONNES. Supposez que vous vouliez par exemple étendre le programme qui place les prénoms dans un tableau, de façon à ce que les noms et les dates de naissance correspondant à chaque prénom puissent être appelées avec le même indice.

Rien de plus facile, direz-vous: il suffit de créer trois tableaux: A$(X) pour les prénoms, B$(X) pour les noms de famille et C$(X) pour les dates de naissance.
Nous pourrions donc de manière générale créer un tableau différent pour chaque rubrique d'une liste de données. Mais nous pourrions alors faire les mêmes remarques qu'avant que nous ne commencions à utiliser les tableaux. Différents tableaux seraient lourds à gérer,

surtout si le nombre de rubriques devenait important. Il existe bien sûr une solution plus pratique :

les tableaux à plusieurs dimensions.

Nous avons dans notre exemple besoin d'un tableaux à deux dimensions pour pouvoir affecter les données de nom et de date de naissance à la même ligne que le prénom correspondant. Il nous faut donc bien un tableau structuré en lignes et en colonnes, comme le montre l'illustration suivante.

NOM DU TABLEAU = A$

	Colonne 1	Colonne 2	Colonne 3
Ligne 1			
Ligne 2			
Ligne 3			
Ligne 4			
Ligne 5			

L'instruction DIM permettant de dimensionner ce tableau sera :

DIM A$(5,3)

Cette instruction génère donc un tableau de 5 lignes sur 3 colonnes. Si vous n'utilisez pas l'instruction DIM mais que vous utilisiez simplement un des éléments de ce tableau, l'ordinateur exécutera automatiquement l'instruction DIM A$(10,10). Vous pouvez donc vous passer de l'instruction DIM tant que votre tableau ne comprend pas plus de 11 lignes ni plus de 11 colonnes. Mais si votre tableau est plus petit, la place perdue du fait du dimensionnement automatique à 11 lignes sur 11 colonnes peut être assez considérable. Il vaut donc mieux, même quand elle n'est pas indispensable, utiliser l'instruction DIM.

Voyons maintenant comment utiliser un tableau à deux dimensions. Supposons par exemple que vous vouliez remplir de données les trois colonnes de la première ligne de notre tableau. Vous pourriez utiliser une ligne de programme telle que celle-ci :

```
40 INPUT"Prenom,Nom,ne le";A$(1,1),A$(1,2),A$(1,3)
```

Cette ligne remplit bien sa fonction puisqu'elle permet d'entrer trois données qu'elle affecte à trois éléments de notre tableau. Mais pour plus de clarté, il vaudrait mieux écrire :

```
.
40 INPUT"Prenom";A$(1,1)
50 INPUT"Nom";A$(1,2)
60 INPUT"Ne le";A$(1,3)
.
```

Vous pourriez bien sûr vous demander pourquoi nous ne faisons pas entrer ces trois données au moyen d'une boucle, maintenant que nous avons un tableau à deux dimensions, qui rend cela parfaitement possible. En l'occurence toutefois, nous ne pourrions utiliser une boucle puisque nous avons besoin de sortir un texte de commentaire correspondant à chacune des trois données à entrer. Nous garderons donc nos trois instructions INPUT.

Mais, comme nous voulons entrer les données de prénom, nom et date de naissance pour 6 personnes différentes, nous pouvons fort bien utiliser une boucle FOR ... NEXT qui nous permette de répéter 6 fois la procédure des lignes 40 à 60 :

```
10 REM Entrer 6 prenoms,
20 REM noms et dates de naissance
30 DIM A$(6,3)
40 FOR I=1 TO 6
50 INPUT"Prenom";A$(I,1)
60 INPUT"Nom";A$(I,2)
70 INPUT"Ne le";A$(I,3)
80 NEXT I:END
```

Vous trouverez des routines semblables dans nombres de petits programmes de gestion de fichier. Les tableaux à plusieurs dimensions peuvent cependant être remplis non seulement à partir du clavier, mais aussi évidemment à partir de lignes de DATA lues par l'instruction READ. Vous connaissez déjà cette technique que nous avons utilisée pour les tableaux à une dimension. Lorsqu'il s'agit d'un tableau à plusieurs dimensions, nous pouvons utiliser des boucles FOR ... NEXT imbriquées. L'exemple suivant vous montre comment on pourrait par exemple remplir de données figurant en lignes de DATA un tableau à deux dimensions d'une taille de 3 lignes sur 4 colonnes :

```
10 REM Charger un tableau a partir d'une ligne DATA
20 DIM X(3,4)
30 FOR Z=1 TO 3
40 FOR S=1 TO 4
50 READ X(Z,S)
60 NEXT S,Z
70 DATA 11,12,13,14,21,22,23,24,31,32,33,34
80 REM Sortie du tableau
90 PRINT"Sortir le tableau (o/n) ?"
```

```
100 A$=INKEY$:IF A$="" THEN 100
110 IF A$<>"o" THEN 150
120 FOR Z=1 TO 3
130 PRINT X(Z,1);X(Z,2);X(Z,3);X(Z,4)
140 NEXT Z
150 END
```

La boucle intérieure (S) remplit de données les divers éléments d'une ligne du tableau. Une fois cette boucle terminée, la boucle extérieure (Z) passe à la ligne suivante, jusqu'à la troisième ligne du tableau. Voici comment on pourrait représenter le remplissage progressif de notre tableau par cet exemple de programme :

TABLEAU A(3,4)

```
11 *  *  *     11 12 *  *     11 12 13 *     11 12 13 14
*  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *
*  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *


11 12 13 14    11 12 13 14    11 12 13 14    11 12 13 14
21 *  *  *     21 22 *  *     21 22 23 *     21 22 23 24
*  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *


11 12 13 14    11 12 13 14    11 12 13 14    11 12 13 14
21 22 23 24    21 22 23 24    21 22 23 24    21 22 23 24
31 *  *  *     31 32 *  *     31 32 33 *     31 32 33 34
```

Si vous voulez faire sortir sur l'écran le tableau que notre programme a créé, vous n'avez qu'à appuyer sur la touche "o". Le tableau sera alors sorti sous la forme que montre l'illustration précédente. La ligne 130 sort en effet en une seule fois les 4 éléments de chaque ligne. Nous n'avons utilisé une boucle que pour sortir les éléments des différentes lignes. Voici toutefois comment nous aurions également pu arriver au même résultat :

```
.
.
.
80 REM Sortie du tableau
90 PRINT"Sortir le tableau (o/n) ?"
100 A$=INKEY$:IF A$="" THEN 100
110 IF A$<>"o" THEN 150
120 FOR Z=1 TO 3
130 FOR S=1 TO 4
140 PRINT A(Z,S);:ZZ=ZZ+1
150 IF ZZ=4 THEN ZZ=0:PRINT:PRINT
160 NEXT S,Z
170 END
```

Cette nouvelle version du programme utilise deux boucles FOR ... NEXT imbriquées. Le point-virgule en ligne 140 fait que les différents éléments d'une même ligne sont affichés l'un à la suite de l'autre. Mais pour que la structure du tableau en lignes et colonnes apparaissent à l'écran, il faut que l'ordinateur passe à la ligne après avoir sorti les 4 éléments d'une même ligne. C'est pourquoi nous avons utilisé un compteur supplémentaire (ZZ) qui compte combien de fois une sortie avec PRINT a été effectuée. La ligne 150 teste si ce compteur ZZ a déjà atteint la valeur 4. Si c'est le cas, le compteur est réinitialisé à 0 puis une instruction PRINT supplémentaire est exécutée dont l'effet est simplement de faire passer la sortie sur écran à la ligne suivante.

Nous avons ajouté une seconde instruction PRINT simplement pour laisser une ligne entre deux lignes de données et rendre la lecture du tableau plus claire.

Maintenant, si vous vouliez pouvoir suivre exactement comment l'ordinateur réalise cette sortie du tableau sur écran, vous pourriez ajouter une temporisation dans votre programme, simplement en entrant la ligne suivante :

```
155 FOR N=1 TO 1000:NEXT N
```

La sortie des données sera alors ralentie par une pause d'environ une seconde entre deux données.

Reprenons maintenant notre exemple de programme pour entrer des noms,prénoms et dates de naissance. Nous avons dit qu'on ne sait pas en général, lorsqu'on réalise un tel programme, pour combien de personnes on aura de données différentes à entrer. On doit par contre toujours essayer de déterminer de façon précise combien de données et lesquelles on voudra pouvoir enregistrer pour chaque personne. Par exemple, on pourra vouloir enregistrer pour chaque personne le nom, le prénom et le numéro de téléphone. Si vous réalisez donc un programme d'agenda téléphonique, vous ne connaissez en général qu'une des deux dimensions nécessaires de votre tableau, celle qui correspond aux nombres de données à entrer pour chaque personne. L'autre dimension, qui correspond donc au nombre de personnes dont vous voudrez pouvoir enregistrer les coordonnées, doit être estimée grossièrement. Par exemple, si vous pensez que votre cercle de relations ne dépasse pas 100 personnes, une instruction DIM X$(120,3) devrait être largement suffisante. Voici un exemple :

```
10 REM Entree de donnees
20 DIM X$(120,3)
30 CLS
```

```
40 Z=Z+1
50 INPUT"Prenom";X$(Z,1)
60 PRINT
70 INPUT"Nom";X$(Z,2)
80 PRINT
90 INPUT"Ne le";X$(Z,3)
100 PRINT
110 PRINT"Voulez-vous entrer "
120 PRINT"d'autres donnees (o/n) ?"
130 A$=INKEY$:IF A$="" THEN 130
140 IF A$="o" THEN 30
150 END
```

Ce problème pourrait être résolu de façon plus élégante, mais l'essentiel est pour nous de bien comprendre le principe. Nous sommes obligé de créer un compteur en ligne 40 pour faire varier l'indice des divers éléments du tableau puisque nous n'utilisons pas de boucle FOR ... NEXT. Les données sont ensuite entrées avec des instructions INPUT et affectées aux éléments correspondants du tableau. Si vous voulez entrer d'autres données, le programme saute en ligne 30, sinon il est terminé. Dans un programme complet de gestion de fichier nous aurions ici au lieu de l'instruction END un retour au menu principal pour permettre à l'utilisateur de sélectionner d'autres points du menu. Vous voyez que nous avons utilisé en ligne 140 une comparaison avec IF ... THEN pour demander à l'utilisateur s'il souhaite entrer d'autres données. Retenez donc bien qu'on ne peut utiliser une boucle FOR ... NEXT que lorsqu'on veut fixer d'avance le nombre de données à entrer. Dans le cas contraire, il faut utiliser IF ... THEN.

Ces exemples devraient suffir pour le moment à vous donner une idée suffisamment claire du maniement des tableaux à plusieurs dimensions. Notez que le CPC peut admettre théoriquement des tableaux ayant jusqu'à 255 dimensions, chaque dimension pouvant aller jusqu'à l'indice 255.

Vous pouvez donc créer des tableaux d'autant de dimensions que vous le voulez et en fait, avant que vous n'atteigniez la limite des 255 dimensions, vous atteindrez la limite imposée par le manque de mémoire. Remarquez d'ailleurs que si nous pouvons nous représenter des tableaux de trois dimensions, sous la forme d'un cube par exemple, nous sommes incapable de nous représenter des tableaux de 4 ou 5 dimensions.

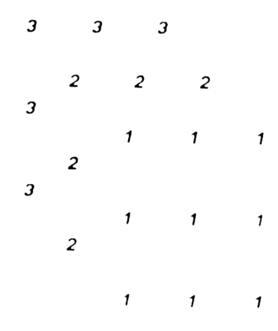
Sans aller jusque là, voici tout de même l'exemple d'un tableau en trois dimensions. Les indices auront la signification suivante :

X = ligne

Y = colonne

Z = profondeur

Nous allons essayer de créer un tableau en trois dimensions qui représente un dé ayant une arête de trois unités. Vous pouvez vous représenter un tel tableau comme trois tableaux à deux dimensions placés l'un derrière l'autre. Avec un peu d'imagination vous pouvez essayer de reconnaître notre dé en perspective sur l'image suivante :



Pour réaliser ce tableau, l'instruction DIM devra avoir le format suivant:

DIM W(X,Y,Z)

Pour notre exemple, nous aurons :

DIM W(3,3,3)

Notre tableau possède ainsi 27 éléments (3*3*3=27) ou plutôt 64 (4*4*4=64) si nous tenions compte de l'élément 0.

Nous vous proposons maintenant l'exercice suivant: écrivez un programme qui remplisse ce tableau de données. Remplissez d'abord tous les éléments d'une ligne, puis toutes les lignes d'une "page" puis toutes les pages du tableau. Essayez de remplir le tableau avec les données qui figurent sur notre illustration. Il est inutile d'essayer d'obtenir un effet de relief. Essayez bien sûr de résoudre ce problème sans regarder la solution.

SOLUTION

```
10 REM Tableau en 3 dimensions
20 DIM W(3,3,3)
30 FOR Z=1 TO 3
40 FOR Y=1 TO 3
50 FOR X=1 TO 3
60 READ W(X,Y,Z)
70 NEXT X,Y,Z
80 DATA 1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3
90 REM Sortie du tableau
100 FOR Z=1 TO 3
110 FOR Y=1 TO 3
120 FOR X=1 TO 3
```

```
130 PRINT W(X,Y,Z);:ZZ=ZZ+1
140 IF ZZ=3 THEN ZZ=0:PRINT
150 NEXT X,Y,Z
160 END
```

L'instruction de dimensionnement en ligne 20 devient de plus en plus indispensable, chaque fois que nous passons à un tableau de dimension supérieure. En effet, alors que nous n'avons ici besoin que de 27 (ou 64) éléments, l'ordinateur créera automatiquement 11*11*11=1331 éléments s'il rencontre une variable à trois indices qui n'a pas été encore dimensionnée. Il exécutera en effet l'instruction DIM W(10,10,10). L'instruction DIM permet donc d'économiser énormément de place en mémoire. Les lignes 30 à 50 vous montrent une imbrication de trois boucles FOR ... NEXT. La boucle extérieure remplit les différentes pages, la boucle du milieu les différentes lignes et la boucle intérieure les différents éléments de chaque ligne. La ligne 70 ferme les trois boucles avec une seule instruction NEXT. Faites toujours bien attention à fermer les boucles dans le bon sens, c'est-à-dire dans le sens exactement contraire au sens d'ouverture.

Ce chapitre sur les tableaux à plusieurs dimensions touche maintenant à sa fin. Il s'agit là d'une technique très importante et très utilisée en BASIC, notamment dans les programmes de gestion de fichier. Bien entendu seuls les tableaux à une ou deux dimensions sont véritablement d'usage courant.

Si vous avez bien assimilé l'utilisation des tableaux, vous pouvez maintenant aborder le chapitre suivant consacré aux sous-programmes qui permettent de gagner du temps de programmation ainsi que de la place en mémoire lorsqu'une même suite d'instructions doit être utilisée plusieurs fois dans le même programme.

4.2 LES SOUS-PROGRAMMES

Un sous-programme est une suite d'instructions ou de lignes d'instructions qui est utilisée plusieurs fois dans un même programme et qu'on va donc pouvoir appeler de différents endroits du programme, chaque fois que cette suite d'instructions devra être exécutée. On place généralement les différents sous-programmes les uns à la suite des autres en début ou en fin de programme. Vous pouvez appeler un sous-programme avec l'instruction :

GOSUB (numéro de ligne)

GOSUB signifie "GO TO SUBROUTINE" (va à la sous-routine). Lorsque le programme rencontre cette instruction, il note le numéro de la ligne où elle figure et d'où il a sauté au sous-programme. Il saute alors au sous-programme et continue l'exécution du programme à partir du début du sous-programme, jusqu'à ce qu'il rencontre l'instruction :

RETURN

L'ordinateur continue alors l'exécution du programme à partir de la première instruction qui figure à la suite de l'instruction GOSUB. Si le programme rencontre une instruction RETURN sans avoir rencontré l'instruction GOSUB auparavant, il s'interrompt en sortant le message d'erreur :

Unexpected RETURN in (numéro de ligne)

Cela signifie que vous ne pouvez appeler un sous-programme qu'avec l'instruction GOSUB. Si vous faites l'erreur de sauter à un sous-programme avec l'instruction GOTO, l'ordinateur ne notera pas où il a rencontré l'instruction GOTO et lorsqu'il rencontrera l'instruction RETURN qui met fin au sous-programme, il ne saura plus où aller et sortira le message d'erreur Unexpected RETURN.

Voici un exemple de ce qu'il ne faut pas faire :

```
110 IF A<1 THEN GOTO 130    !!! E R R E U R ! ! !
120 GOTO 90
130 REM SOUS-PROGRAMME
140 A=A+1
150 RETURN
```

Si A est plus petit que 1, le programme sautera ici en 130. Il exécutera la ligne 140 puis lorsqu'il rencontrera RETURN, il sortira le message d'erreur :

Unexpected RETURN in 150

et interrompra le programme. Il aurait fallu écrire en ligne 110 :

```
110 IF A<1 THEN GOSUB 130
```

Le programme suivant illustre une autre erreur favorite des programmeurs de sous-programmes, qui de plus est assez difficile à dépister :

```
10 REM ERREUR AVEC UN SOUS-PROGRAMME
20 CLS:PRINT
30 PRINT CHR$(24);Z;CHR$(24)
40 GOSUB 70
50 Z=Z+1:GOTO 20
60 END
70 REM SOUS-PROGRAMME
80 FOR I=1 TO 25
90 PRINT I;
100 IF I >= 15 THEN 50
110 NEXT I
120 RETURN
```

Si vous lancez ce programme, vous constaterez qu'après avoir parcouru 17 fois le sous-programme, le déroulement du programme sera interrompu avec le message d'erreur : **Memory full in 100**

L'erreur ne vient pas ici de l'appel mais de la sortie du sous-programme. Le programme commence en effet par vider l'écran puis par constituer une ligne d'espaces (ligne 20). La valeur actuelle du compteur Z qui compte le nombre d'appels du sous-programme est ensuite sortie en inversion vidéo (CHR$(24)). La ligne 40 appelle alors le sous-programme avec GOSUB 70. Le programme saute au sous-programme en ligne 70 et commence par exécuter la boucle FOR ... NEXT. La ligne 90 sort la valeur de la variable-compteur I.

La ligne 100 contient ensuite 2 erreurs en une seule: premièrement il ne faut jamais quitter une boucle FOR ... NEXT avec GOTO sous peine de provoquer des problèmes dans la gestion de ces boucles par l'ordinateur. Deuxièmement et surtout, il ne faut pas sortir d'un sous-programme autrement qu'en utilisant l'instruction RETURN. Or la ligne 100 saute du sous-programme à la ligne 50 si I est supérieur ou égal à 15. Il aurait donc fallu écrire THEN RETURN au lieu de THEN 50.

Vous pouvez bien sûr utiliser autant d'instructions GOTO que vous voulez à l'intérieur d'un sous-programme mais vous ne devez pas quitter un sous-programme ni entrer dans un sous-programme avec l'instruction GOTO. Ce type d'erreur est d'ailleurs facile à éviter avec un organigramme clair.

Essayons maintenant d'appliquer la technique des sous-programmes à un exemple concret. Reprenons pour cela le programme de cours de calcul du chapitre 3.3.1. Si vous étudiez le listing de ce programme, vous constaterez vite que certaines parties du programme se répètent plusieurs fois. On pourrait donc tout à fait remplacer ces parties du programme par un sous-programme. Voici notamment les lignes qui se répètent souvent :

```
230 CLS
240 PRINT TAB(10)"Entrez le plus grand nombre"
250 PRINT
260 PRINT TAB(10)"pour l'addition."
270 PRINT
290 PRINT TAB(10);:INPUT"Plus grand nombre";GR
299 REM
300 REM GENERATION DE NOMBRES ALEATOIRES
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM CALCUL DU RESULTAT
331 REM
340 ER=A1+A2
350 CLS
360 PRINT
370 PRINT A1 "+" A2 "=";
380 INPUT ES
390 IF ES=ER THEN PRINT:PRINT TAB(10)
    "C'EST JUSTE!":F=0:GOTO 450
400 PRINT:PRINT  TAB(10)"C'EST FAUX!"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5) "Le résultat est";ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(5) "Encore un exercice O/N";
490 INPUT A$
500 IF A$="O" THEN F=0:GOTO 300
510 GOTO 10
```

Ces lignes de programme qui figurent à 4 reprises dans notre programme pourraient être regroupées en différents blocs. Un premier bloc pourrait être constitué par les lignes 230 à 290 car l'entrée d'un nombre maximum est demandée pour chaque type de calcul. Le seul problème est que le message demandant à l'utilisateur d'entrer le nombre maximum doit pouvoir être adapté en fonction de chaque type de calcul. Le message :

Entrez le plus grand nombre

pour l'addition

doit donc pouvoir être modifié par le sous-programme en fonction du type de calcul.

Souvenons-nous par ailleurs que le programme saute au type de calcul choisi par l'utilisateur avec l'instruction ON X GOTO. Nous pourrions donc utiliser cet X comme indice du type de calcul. Nous pouvons ainsi créer en début de programme un tableau qui contienne les termes d'addition, soustraction, division et multiplication, dans l'ordre dans lequel ces différentes opérations figurent dans le menu. Nous pourrions même alors faire éditer le menu grâce à ce tableau. Voici comment se présenterait le début du programme ainsi modifié :

```
10 REM   ******************************
20 REM   * DEBUT DU PROGRAMME *
30 REM   ******************************
40 REM
50 DIM calcul$(4),operateur$(4)
60 FOR I=1 TO 4
70 READ calcul$(I),operateur$(I)
80 NEXT I
90 DATA l'addition,+,la soustraction,-,division,/,multiplication,*
100 GOTO 580
```

.
.
.

La ligne 50 génère deux tableaux calcul$ et operateur$ (n'oubliez pas que vous pouvez utiliser avec le CPC des noms de variables ou tableaux très explicites). La boucle FOR ... NEXT en ligne 60 charge données en DATA dans les deux tableaux en affectant à calcul$ les noms de type de calcul et à operateur$ les signes correspondant à chaque type d'opération. Nous verrons plus tard pourquoi le programme saute de la ligne 100 à la ligne 580. Voyons maintenant comment nous pouvons utiliser les tableaux que nous venons de créer pour sortir le menu sur l'écran. Voici donc comment se présente maintenant le menu :

```
.
.
.
570 REM ****************
580 REM *    MENU      *
590 REM ****************
600 CLS:F=0
610 PRINT
620 PRINT TAB(12)"COURS DE CALCUL"
630 PRINT:PRINT
640 PRINT TAB(12)"Choisissez:"
650 PRINT
660 PRINT TAB(12)"1 pour "calcul$(1)
670 PRINT
680 PRINT TAB(12)"2 pour "calcul$(2)
690 PRINT
700 PRINT TAB(12)"3 pour "calcul$(3)
710 PRINT
720 PRINT TAB(12)"4 pour "calcul$(4)
730 PRINT
```

```
740 PRINT TAB(12)"5 pour FIN"
750 PRINT
760 PRINT TAB(12)"Quel nombre"
770 E$=INKEY$:IF E$="" THEN 770
780 IF VAL(E$)<1 OR VAL(E$)>5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
.
.
.
```

Nous aurions aussi bien pu créer le menu à l'aide d'une boucle, comme dans l'exemple ci-dessous :

```
.
.
.
660 FOR I=1 TO 4
670 PRINT TAB(12);I;"pour ";calcul$(I)
680 PRINT
690 NEXT I
.
.
.
```

Mais nous y avons renoncé pour conserver une présentation plus claire au programme.

Dans ce nouveau menu, la ligne 770 lit avec INKEY$ un caractère et la ligne 780 teste si ce caractère est "recevable". Si c'est bien le cas, la valeur numérique de ce caractère est calculée avec VAL(E$) et affectée à la variable P. Le programme saute ensuite à différentes parties du programme, en fonction de la valeur de P. Pour l'addition par exemple, vous entrerez 1, P vaudra 1 et le programme sautera à la ligne 800. C'est en effet en 800 que commence la partie du programme, le "module" de l'addition. Voyons comment il se présente :

```
.
.
.
800 REM *************
810 REM  ADDITION
820 REM *************
830 GOSUB 110
840 GOSUB 310
850 ER=A1+A2
860 GOSUB 390
870 IF A$="o" THEN 840
880 GOTO 580
.
.
.
```

La ligne 830 appelle maintenant le premier sous-programme. Ce sous-programme permet l'entrée du nombre le plus grand devant être utilisé dans le calcul. Voici comment il se présente maintenant :

```
.
.
.
110 REM **************************
120 REM * SOUS-PROGRAMMES *
130 REM **************************
140 REM ******************************
150 REM * Entree du plus grand nombre   *
160 REM ******************************
170 CLS:A$="":B$=""
180 PRINT TAB(10)"Entrez le plus grand nombre"
190 PRINT
200 PRINT TAB(10)"pour "calcul$(P)
210 PRINT
220 PRINT TAB(10)"Plus grand nombre ?";
```

```
230 FOR I=1 TO 3
240 A$=INKEY$:IF A$="" THEN 240
250 IF ASC(A$)<48 OR ASC(A$)>57 THEN 240
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=val(B$)
290 RETURN
.
.
.
```

L'instruction CLS en ligne 170 vous est certainement déjà devenue familière. Mais la deuxième partie de cette ligne mérite un commentaire car elle est typique pour un sous-programme. Les variables A$ et B$ sont en effet annulées, on leur affecte une chaîne vide pour être sûr que le dispositif de la ligne 260 fonctionnera bien, même si ce sous-programme est appelé à plusieurs reprises, c'est-à-dire pour plusieurs types d'opérations différentes. En effet B$ reçoit en 260 le nombre que vous entrez. Une fois que le programme sortira du sous-programme, B$ gardera ce nombre intact. Si donc le programme saute à nouveau au même sous-programme pour un autre type d'opération, le nombre que vous entrerez cette fois ira s'ajouter aux chiffres que B$ aura reçu la première fois. Il faut donc toujours penser à réinitialiser les variables de ce type au début d'un sous-programme.

Selon une technique qui nous est maintenant familière, nous entrons les chiffres composant le plus grand nombre en ligne 240 et la ligne 250 contrôle qu'il ne s'agit que de chiffres entre 0et 9. PRINT A$; en ligne 260 sort le caractère que l'utilisateur vient d'entrer et lui permet ainsi de voir ce qu'il entre. Notre programme n'admet que des nombres de trois chiffres. Si vous voulez entrer un nombre de deux chiffres, vous devez entrer d'abord un 0 puis vos deux chiffres.

La ligne 200 vous montre pourquoi nous avons créé un tableau contenant les noms des opérations : cela nous permet en effet d'adapter le message demandant l'entrée du plus grand nombre en fonction du type d'opération fourni par P.

La ligne 840 du programme d'addition appelle alors le sous-programme de génération de nombres aléatoires qui est le sous-programme le plus petit et le plus simple :

```
.
.
300 REM
310 REM GENERATION DE NOMBRES ALEATOIRES
320 REM
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
.
.
```

Voici maintenant le sous-programme de création de l'exercice :

```
360 REM
370 REM CREATION D'UN EXERCICE
380 REM
390 CLS
400 PRINT
410 PRINT A1;operateur$(P);A2 "=";
420 INPUT ES
430 IF ES=ER THEN PRINT:PRINT TAB(10)
    "C'EST JUSTE!":F=0:GOTO 500
440 PRINT:PRINT TAB(10)"C'EST FAUX!"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
470 IF F<=2 THEN 390
480 PRINT
```

```
490 FOR I=0 TO 2000:NEXT I
500 PRINT TAB(5) "Le résultat est";ER
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5) "Encore un exercice O/N";
550 INPUT A$
560 RETURN
.
.
```

Nous nous arrêterons ici uniquement sur les lignes méritant un commentaire particulier. D'abord la ligne 410: comme nous l'avons fait dans le sous-programme d'entrée du plus grand nombre, nous utilisons ici un tableau pour créer un message pour l'utilisateur. Ce message pourrait être en effet symbolisé ainsi :

A1 + (ou - ou / ou *) A2 = ?

L'opérateur qui convient est sélectionné en fonction du type d'opération P. Nous avons donc généralisé la formulation du message à l'utilisateur pour lui présenter un exercice. Ce faisant nous avons été obligé de fixer une fois pour toute les variables opérandes: le premier opérande sera toujours A1, le second toujours A2. Ceci va nous conduire à modifier quelque peu la partie du programme consacrée à la division puisque la formule d'exercice choisie pour la division n'était pas A1 / A2 mais ER / A1.
Vous voyez que les sous-programmes peuvent entraîner et entraînent effectivement souvent des adaptations du programme. Le programme doit être pensé dès le départ de façon à permettre d'isoler un maximum de tâches générales qui pourront être regroupées dans des sous-programmes. Il faut ensuite se demander pour chaque partie du programme appelant un sous-programme quelle préparation doivent subir telles ou telles variables pour que le sous-programme donne bien les résultats attendus.

Nous en avons fini avec l'étude de la partie sous-programmes de notre programme. Reste un petit problème que nous voudrions évoquer. Vous avez certainement remarqué que nous avons placé les sous-programmes au début de notre nouvelle version du programme alors que nombre d'ouvrages recommandent de placer les sous-programmes en fin de programme. En effet, si vous voulez constituer une bibliothèque de sous-programmes, il peut être très pratique d'en placer en 50000, en 55000 etc... et de les ajouter ensuite à la suite de vos programmes avec l'instruction MERGE.

Voyons cependant les arguments qui parlent en faveur de sous-programmes placés en début de programme : on peut objecter que peu importe finalement que ajoutiez un sous-programme à votre programme principal ou que vous ajoutiez votre programme principal à la suite d'un sous-programme. D'autant que l'instruction RENUM vous permet de jongler avec les numéros de ligne de votre sous-programme ou de votre programme. Jusqu'ici donc, match nul.

Mais si on étudie le fonctionnement de votre ordinateur, on s'aperçoit que lorsque vous appelez un sous-programme, l'ordinateur cherche le numéro de la ligne à laquelle il doit sauter en examinant le programme du début. Si donc vous placez les sous-programmes au début du programme, vous diminuez son temps de recherche et donc aussi le temps d'exécution du programme.

Voyons maintenant quelles adaptations le reste du programme va devoir subir pour s'adapter à l'utilisation des sous-programmes. Il s'agit en effet de préparer les variables A1 et A2 pour l'exercice dont nous avons vu qu'il se présentera toujours sous la forme A1 (opérateur) A2 = ?. En fait seules la soustraction et la division nécessitent des adaptations particulières.

En ce qui concerne la soustraction, il faut que A1 soit toujours plus grand que A2.

C'est ce que garantissait en fait déjà la ligne 940 qui échange entre elles les valeurs de A1 et A2 lorsque A1 est inférieur à A2. En fait cette adaptation existait déjà dans la première version de notre programme.

Pour la division par contre nous allons être obligé de modifier quelque peu la logique antérieure du programme. L'exercice de division se présentait en effet ainsi :
ER / A1 = ?

Or nous avons réalisé un sous-programme général d'exercice auquel la division doit également se plier. L'exercice de division doit donc obligatoirement pouvoir se présenter ainsi :
A1 / A2 = ?

Et ER doit être le résultat de cet exercice. C'est ce qui sera réalisé avec les lignes supplémentaires suivantes :

```
.
.
1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
.
.
```

Nous calculons toujours le résultat grâce à une multiplication mais nous réalisons en ligne 1050 un échange entre les deux variables A1 et A2 de façon à ce qu'elles soient correctement préparées pour le sous-programme "création d'un exercice". Nous utilisons pour ce faire à nouveau la technique du stockage provisoire : I reçoit provisoirement la valeur de la variable ER. ER reçoit la valeur de A1 qui reçoit celle de I, donc la valeur antérieure de ER.

Nos trois variables A1, A2 et ER sont maintenant prêtes et le sous-programme de création d'un exercice donnera bien les résultats escomptés. Voici maintenant le listing complet de la nouvelle version de notre programme :

```
10 REM     **********************
20 REM     * DEBUT DE PROGRAMME *
30 REM     **********************
40 REM
50 DIM calcul$(4),operateur$(4)
60 FOR i=1 TO 4
70 READ calcul$(i),operateur$(i)
80 NEXT i
90 DATA l'addition,+,la soustraction,-,division,/,multiplica
tion,*
100 GOTO 580
110 REM *******************
120 REM * SOUS-PROGRAMMES *
130 REM *******************
140 REM ---------------------------
150 REM Entree du plus grand nombre
160 REM ---------------------------
170 CLS:a$="":b$=""
180 PRINT TAB(10)"Entrez le plus grand nombre"
190 PRINT
200 PRINT TAB(10)"pour "calcul$(p)
210 PRINT
220 PRINT TAB(10)"Plus grand nombre ?";
230 FOR i=1 TO 3
240 a$=INKEY$:IF a$="" THEN 240
250  IF ASC(a$)<48 OR ASC(a$)>57 THEN 240
260 b$=b$+a$:PRINT a$;
270 NEXT i
280 gr=VAL(b$)
290 RETURN
300 REM --------------------------------
310 REM Generation de nombres aleatoires
320 REM --------------------------------
330 a1=INT(gr*RND(1))+1
340 a2=INT(gr*RND(1))+1
350 RETURN
360 REM ----------------------
370 REM Creation d'un exercice
380 REM ----------------------
```

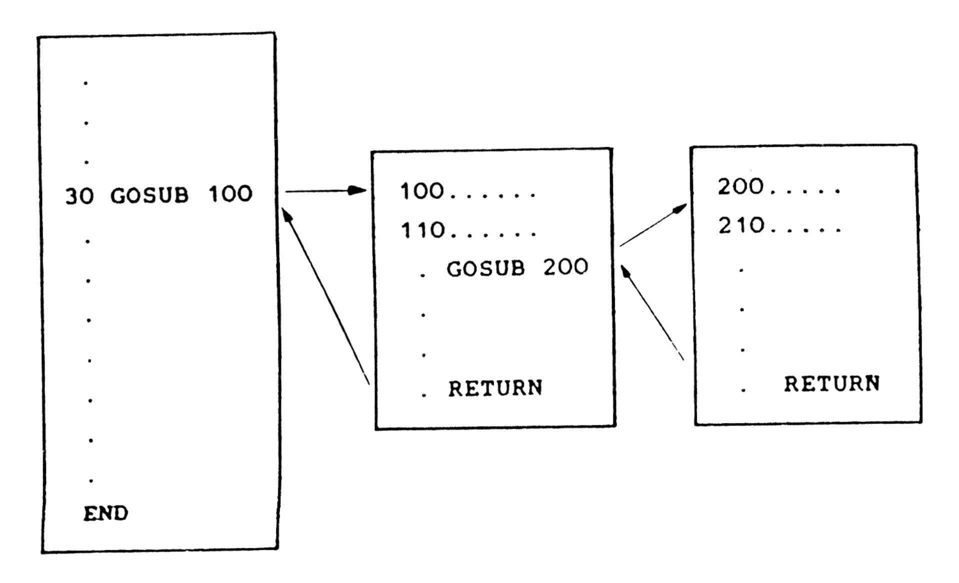
```
390 CLS
400 PRINT
410 PRINT a1;operateur$(p);a2 "=";
420 INPUT es
430 IF es=er THEN PRINT:PRINT TAB(10)"C'est juste":f=0:GOTO
500
440 PRINT:PRINT TAB(10)"C'est faux"
450 FOR i=0 TO 2000
460 NEXT i:f=f+1
470 IF f<=2 THEN 390
480 PRINT
490 FOR i=1 TO 2000:NEXT i
500 PRINT TAB(5) "Le resultat est";er
510 f=0
520 FOR i=0 TO 3000
530 NEXT i: PRINT
540 PRINT TAB(5)"Encore un exercice O/N";
550 INPUT a$
560 RETURN
570 REM ********
580 REM * MENU *
590 REM ********
600 CLS
610 f=o
615 PRINT TAB(12)"COURS DE CALCUL"
620 PRINT:PRINT TAB(12)"Choisissez:"
630 PRINT
640 PRINT TAB(12)"1 Pour "calcul$(1)
650 PRINT
660 PRINT TAB(12)"2 Pour "calcul$(2)
670 PRINT
680 PRINT TAB(12)"3 Pour "calcul$(3)
690 PRINT
700 PRINT TAB(12)"4 Pour "calcul$(4)
720 PRINT
740 PRINT TAB(12)"5 Pour FIN"
750 PRINT
760 PRINT TAB(12)"Quel nombre"
770 e$=INKEY$:IF e$="" THEN 770
780 IF VAL(e$)<1 OR VAL(e$)>5 THEN 770
```

```
790 p=VAL(e$):ON p GOTO 800,890,990,1090,1180
800 REM ************
810 REM * ADDITION *
820 REM ************
830 GOSUB 110
840 GOSUB 310
850 er=a1+a2
860 GOSUB 390
870 IF a$="o" THEN 840
880 GOTO 580
890 REM ****************
900 REM * SOUSTRACTION *
910 REM ****************
920 GOSUB 110
930 GOSUB 310
940 IF a1<a2 THEN i=a1:a1=a2:a2=i
950 er=a1-a2
960 GOSUB 390
970 IF a$="o" THEN 930
980 GOTO 580
990 REM  ************
1000 REM * DIVISION *
1010 REM ************
1020 GOSUB 110
1030 GOSUB 310
1040 er=a1*a2
1050 i=er:er=a1:a1=i
1060 GOSUB 390
1070 IF a$="o" THEN 1030
1080 GOTO 580
1090 REM ******************
1100 REM * MULTIPLICATION *
1110 REM ******************
1120 GOSUB 110
1130 GOSUB 310
1140 er=a1*a2
1150 GOSUB 390
1160 IF a$="o" THEN 1130
1170 GOTO 580
1180 REM *******
```

```
1190 REM * FIN *
1200 REM *******
1210 CLS
1220 END
```

Nous avons donc ainsi économisé environ 43 lignes de programme grâce à l'utilisation de trois sous-programmes et bien que nous ayons ajouté des lignes de commentaires REM ! La technique des sous-programmes n'est pas seulement pratique parce qu'elle permet de gagner du temps lors de la programmation mais elle permet donc également de gagner de la place en mémoire. Vous avez certainement compris maintenant pourquoi la ligne 100 contient l'instruction GOTO 580 : le programme doit en effet sauter directement au menu, en sautant par dessus les sous-programmes. Nous avons déjà expliqué pour quelles raisons nous pensons qu'il est préférable de placer les sous-programmes en début de programme.

Les sous-programmes, comme les boucles, peuvent être imbriqués : un sous-programme peut appeler un autre sous-programme, ce que l'on pourrait symboliser ainsi :



Le programme rencontrera donc l'instruction GOSUB en ligne 30. Il sautera alors au sous-programme 1 en ligne 100. Le sous-programme 1 appellera ensuite le sous-programme 2 avec GOSUB 200. Le sous-programme 2 sera exécuté jusqu'à l'instruction RETURN. Le programme sortira alors du sous-programme 2 pour retourner au sous-programme 1 qu'il exécutera à partir de l'instruction qui suit directement l'instruction GOSUB. Le sous-programme 1 sera également exécuté jusqu'à la prochaine instruction RETURN, après quoi le programme retournera au programme principal qu'il continuera d'exécuter à partir de l'instruction qui suit l'instruction GOSUB.

Les sous-programmes peuvent également être appelés avec une instruction qui équivaut pour les sous-programmes à l'instruction ON ... GOTO, l'instruction ON ... GOSUB :

ON P GOSUB 800,890,990

Suivant la valeur de P, le programme sautera au sous-programme 800, 890 ou 990. Si P vaut 0 ou plus de 3, le programme exécutera l'instruction placée à la suite de l'instruction ON ... GOSUB. Notez simplement qu'une fois le sous-programme exécuté, le programme reprendra l'exécution du programme principal à partir de l'instruction qui suit l'instruction ON ... GOSUB.

Nous espérons que l'explication détaillée de notre exemple de cours de calcul vous aura permis de bien comprendre la technique des sous-programmes. Pour vous permettre de vous en assurer, voici un petit exercice.
Nous n'avons pas en effet épuisé dans notre dernière version du cours de calcul les possibilités de la technique des sous-programmes. Essayez donc de regrouper le plus grand nombre possible de lignes dans des sous-programmes. Bien sûr vous devrez adapter quelque peu le programme principal. Nous ne vous demandons pas de placer ces nouveaux sous-programmes en début de programme car cela vous obligerait à trop de travail d'écriture. Essayez de trouver la solution par vous-même. Bonne chance !

SOLUTION

```
.
.
.
780 IF VAL(E$)<1 OR VAL(E$)>5 THEN 770
790 P=VAL(E$)
800 IF P=5 THEN 1100
810 GOSUB 110
820 GOSUB 310
830 ON P GOSUB 880,930,990,1050
840 GOSUB 390
850 IF A$="o" THEN 820
860 GOTO 580
870 REM ***************
880 REM  ADDITION
890 REM ***************
900 ER=A1+A2
910 RETURN
920 REM ******************
930 REM  SOUSTRACTION
940 REM ******************
950 IF A1 < A2 THEN I=A1:A1=A2:A2=I
960 ER=A1-A2
970 RETURN
980 REM  **********
990 REM   DIVISION
1000 REM **********
1010 ER=A1*A2
1020 I=ER:ER=A1:A1=I
1030 RETURN
```

```
1040 REM *********************
1050 REM  MULTIPLICATION
1060 REM *********************
1070 ER=A1*A2
1080 RETURN
1090 REM *******
1100 REM * FIN *
1110 REM *******
1120 CLS
1130 END
```

Nous espérons que vous avez trouvé la solution sans peine. Il y avait en effet 5 lignes de programme qui se répétaient dans les parites du programme consacrées aux différentes opérations :

```
830 GOSUB 110
840 GOSUB 310
860 GOSUB 390
870 IF A$="o" THEN 840
880 GOTO 580
```

Les parties du programme pour chaque type d'opération ne se distinguent donc entre elles que par le calcul proprement dit. C'est pourquoi les lignes 830, 840, 860, 870 et 880 ont été placées à la suite du menu. Un test spécial sur T=5 a été instauré car la fin du programme n'est pas un sous-programme. Les calculs pour les types d'opération ont été alors écrits comme des sous-programmes qui seront appelés avec l'instruction :

ON P GOSUB

Nous avons ainsi économisé 9 lignes de programme supplémentaires.

Résumons maintenant les points les plus importants à retenir en ce qui concerne la technique des sous-programmes :

1. Les sous-programmes permettent de regrouper des parties de programme qui se répétent fréquemment.

2. Les sous-programmes sont appelés avec GOSUB et terminés avec RETURN. Ils ne doivent jamais être appelés ou quittés avec l'instruction GOTO. L'instruction GOTO doit être utilisée soit totalement en dehors d'un sous-programme, soit totalement à l'intérieur d'un sous-programme.

3. Les sous-programmes peuvent être imbriqués: un sous-programme 1 peut appeler un sous-programme 2 qui lui-même peut appeler un sous-programme 3, etc... La seule limite tient à la capacité mémoire de votre ordinateur. Faites attention à ce qu'il y ait bien toujours une instruction RETURN qui corresponde à chaque instruction GOSUB.

4. Les sous-programmes peuvent être appelés également avec l'instruction ON X GOSUB.

Il nous reste, pour être complet, à expliquer trois dernières instructions qui jouent également un rôle dans l'utilisation des sous-programmes :

ON BREAK GOSUB

Cette instruction ne provoquera de branchement à un sous-programme que lorsque vous aurez appuyé 2 fois sur la touche ESC. Cette instruction permet donc d'inhiber la fonction de la touche ESC dans certaines parties de votre programme qui ne pourra donc plus être interrompu au moyen de cette touche. Voici un exemple :

```
10 CLS
20 ON BREAK GOSUB 100
30 LOCATE 3,5
```

```
40 PRINT "Appuyez deux fois sur la touche ESC."
50 FOR I=1 TO 100:NEXT I
60 LOCATE 3,5
70 PRINT CHR$(24); "Appuyez deux fois sur la touche ESC." ;
  CHR$(24)
80 FOR I=1 TO 100:NEXT I
90 GOTO 30
100 CLS
110 LOCATE 2,12
120 PRINT "La touche ESC a été actionnée deux fois."
130 FOR I=1 TO 1000:NEXT I
140 CLS:RETURN
```

Si vous lancez ce programme, vous ne pourrez l'arrêter qu'en provoquant un RESET général de l'ordinateur, c'est-à-dire qu'en le remettant dans son état initial après allumage avec les touches CTRL, SHIFT et CTRL. Il existe cependant une instruction qui complète l'instruction ON BREAK GOSUB :

ON BREAK STOP

Modifiez maintenant notre programme de la façon suivante :

```
140 ON BREAK STOP
150 CLS:RETURN
```

Si vous lancez cette nouvelle version du programme, le premier double ESC provoquera un saut du programme au sous-programme mais le second double ESC interrompra le programme.

Voici enfin l'instruction : ON SQ GOSUB ... , ... , ...

qui est utilisée essentiellement dans la programmation du son. Le programme saute à une des lignes suivant le GOSUB en fonction de la valeur de la variable système SQ.

4.2.1 SOUS-PROGRAMMES AVEC AFTER ET EVERY

Le CPC dispose de deux autres instructions permettant d'appeler des sous-programmes. Mais à la différence des instructions que nous avons vues jusqu'ici, ces instructions appellent un sous-programme en fonction d'un certain facteur de temps :

AFTER X,Y GOSUB (numéro de ligne)

Cette instruction sautera au sous-programme lorsqu'un délai indiqué par X et Y se sera écoulé. X indique la durée de ce délai en 50èmes de seconde et Y indique le timer (l'horloge) utilisée. Y doit être compris entre 0 et 3, la valeur standard étant 0. Lorsque le délai se sera écoulé, le programme sautera au sous-programme qu'il exécutera normalement, jusqu'à ce qu'il rencontre l'instruction RETURN. Le programme poursuivra alors l'exécution du programme principal à partir de l'endroit, quel qu'il soit, où il se trouvait lorsqu'il a sauté au sous-programme. Voici un programme qui sort par exemple un message après un délai de 20 secondes, pendant lesquelles vous pouvez voir le temps indiqué dans l'angle supérieur gauche de l'écran :

```
10 TEMPS=INT(TIME/300)
20 CLS
30 AFTER 1000 GOSUB 70
40 TEMPS1=INT(TIME/300)-TEMPS
50 PRINT CHR$(30);TEMPS1;"SEC"
60 GOTO 40
70 REM Sous-programme
80 CLS
90 LOCATE 5,14
100 PRINT TEMPS1;" secondes se sont écoulées."
110 RETURN
```

Vous pouvez interrompre ce programme avec la touche ESC. Si vous voulez maintenant que le programme sorte un tel message toutes les 20 secondes, utilisez l'instruction :

EVERY X,Y GOSUB (numéro de ligne)

Le sous-programme sera alors exécuté après chaque intervalle dont la durée est indiquée par X et Y. Modifions notre programme en entrant la ligne suivante :

30 EVERY 1000 GOSUB 70

Si vous lancez cette nouvelle version du programme, vous constaterez qu'un message sera maintenant sorti toutes les 20 secondes. En liaison avec cette instruction, vous pouvez utiliser l'instruction :

REMAIN

Si vous entrez :

PRINT REMAIN (Y)

où Y est le numéro du timer et doit être compris entre 0 et 3, vous obtiendrez en réponse le nombre d'unités de temps restant encore à s'écouler sur le timer Y. Si vous utilisez d'autres valeurs, vous obtiendrez le message d'erreur:

Improper argument

4.3 LA TECHNIQUE DES MENUS

Maintenant que vous avez avancé dans la maîtrise de la programmation en BASIC, vous voudrez certainement un jour écrire vous-même des programmes d'une taille importante, soit pour vous faciliter une tâche donnée, soit pour commercialiser ces programmes. En tout cas, il sera important que vos programmes soient commodes et agréables à utiliser.

Il importe donc que l'utilisateur éventuel n'ait pas à chercher trop longtemps sur quelle touche il doit appuyer pour obtenir tel ou tel résultat ou fonction. Bien sûr un programme de taille importante doit être accompagné d'un manuel mais il ne faut pas non plus que l'utilisateur ait besoin de recourir sans cesse au manuel pour "deviner" comment il va pouvoir mettre en route telle ou telle opération.

Nous n'allons pas vous demander d'écrire un manuel mais nous voudrions simplement que vous maîtrisiez bien les principes de la technique des menus.

Comme nous l'avons déjà indiqué pour notre programme de cours de calcul, un menu est une "page" d'écran qui expose les différents points du programme, les différentes opérations que celui-ci permet. L'utilisateur peut alors sélectionner un de ces points du programme en appuyant sur la touche qui est associée dans le menu au point du programme qu'il a choisi. La présentation du menu est bien sûr laissée à l'imagination du programmeur mais il convient toutefois de toujours essayer de la rendre la plus claire possible. Il est toujours bien venu d'utiliser des séparations visuelles constituées de différentes suites de caractères, pourvu que ces séparations ne surchargent pas l'écran.

Nous allons maintenant voir par un exemple concret comment on crée un menu. Nous allons essayer de créer un programme de tables mathématiques que l'utilisateur pourra consulter.

Supposons que nous ayons besoin des opérations suivantes :

racine carrée
sinus
cosinus
logarithme naturel
logarithme de base 10

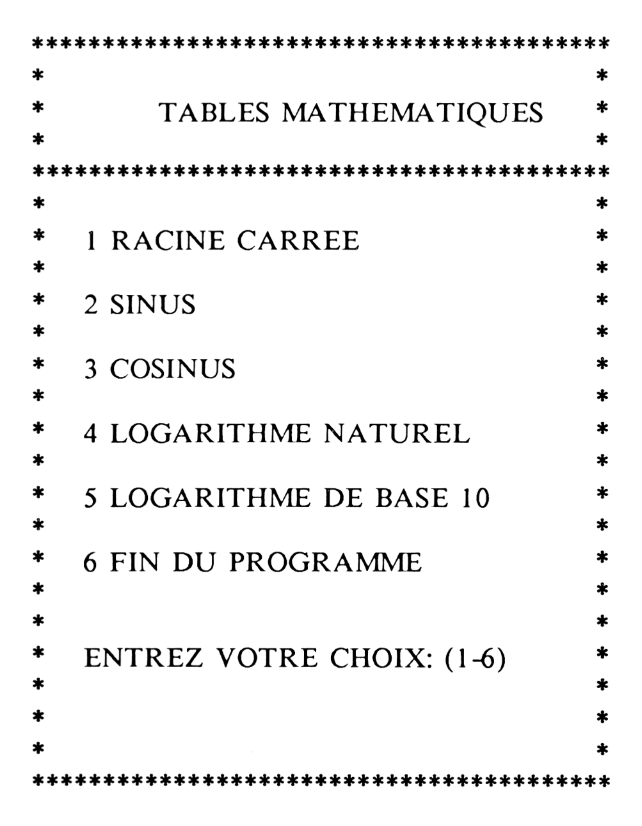
Notre programme doit donc pouvoir exécuter un de ces 5 calculs au choix.

Il manque encore un point à notre menu : la fin du programme, de façon à ce que l'utilisateur puisse mettre fin au programme dans le cadre de celui-ci, sans utiliser la touche ESC ni éteindre l'ordinateur. Notre menu offrira donc 6 options. Il faudra d'autre part que notre menu envoie un message à l'utilisateur pour l'inviter à entrer un chiffre ou une lettre comme par exemple :

Entrez votre choix : (1-6) ?

Il ne nous manque plus qu'un titre et un cadre pour améliorer la présentation du menu. Les couleurs de l'écran devront être conservées et il serait bien que le titre du programme demeure à l'écran, quelle que soit la partie du programme exécutée. Il nous faudra donc utiliser un sous-programme pour le titre.

Voici comment notre menu devra apparaître sur l'écran :



```
***************************************
*                                     *
*         TABLES MATHEMATIQUES        *
*                                     *
***************************************
*                                     *
*   1 RACINE CARREE                   *
*                                     *
*   2 SINUS                           *
*                                     *
*   3 COSINUS                         *
*                                     *
*   4 LOGARITHME NATUREL              *
*                                     *
*   5 LOGARITHME DE BASE 10           *
*                                     *
*   6 FIN DU PROGRAMME                *
*                                     *
*                                     *
*   ENTREZ VOTRE CHOIX: (1-6)         *
*                                     *
*                                     *
*                                     *
***************************************
```

La dernière ligne doit en même temps recevoir l'entrée du chiffre choisi. Utilisons tout d'abord l'instruction INPUT pour l'entrée et nous verrons plus tard si nous pouvons employer ici l'instruction INKEY$. Voici comment se présente le programme permettant de créer ce menu :

```
10 REM *********************
20 REM * DEBUT DE PROGRAMME *
30 REM *********************
40 REM
50 CLS
60 DIM m$(6)
70 FOR i=1 TO 6
80 READ m$(i):NEXT i
90 DATA " 1 Racine carree"
100 DATA " 2 Sinus"
110 DATA " 3 Cosinus"
120 DATA " 4 Logarithme naturel"
130 DATA " 5 Logarithme de base 10"
140 DATA " 6 Fin de programme"
150 GOTO 330
160 REM ****************
170 REM * SOUS-PROGRAMME *
180 REM ****************
190 REM
200 REM *************
210 REM * LIGNE-TITRE *
220 REM *************
230 CLS
240 PRINT STRING$(40,"*");
250 PRINT"*                                      *";
260 PRINT"*          TABLE MATHEMATIQUES         *";
270 PRINT"*                                      *";
280 PRINT STRING$(40,"*");
290 RETURN
300 REM ********
310 REM * MENU *
320 REM ********
330 GOSUB 230
340 FOR i=1 TO 19
350 PRINT"*                                      *";
360 NEXT i
370 PRINT STRING$(40,"*");
380 PRINT CHR$(30);
390 FOR i=1 TO 5:PRINT:NEXT i
```

```
400 FOR i=1 TO 6
410 PRINT CHR$(10);CHR$(9);m$(i)
420 NEXT i
430 PRINT
440 PRINT CHR$(10);CHR$(9);
450 PRINT" Entrez votre choix (1-6)";
460 INPUT w$
```

La ligne 50 vide l'écran et les lignes 60 à 80 créent le tableau M$ avec les données figurant dans les lignes DATA 90 à 140. Le programme saute alors par dessus les sous-programmes à la ligne 330. Il saute alors au sous-programme en ligne 230 qui crée la ligne-titre. Les lignes 240 et 280 créent les lignes d'étoiles qui entoureront la ligne-titre.

Le programme quitte alors le sous-programme pour retourner en ligne 340 où la boucle (340 à 360) crée le menu. La ligne 370 sort la dernière ligne du menu et la ligne 380 replace le curseur dans l'angle supérieur gauche de l'écran. La ligne 390 sort 5 lignes blanches pour que les points du menu n'effacent pas la ligne-titre. Les lignes 400 à 420 sortent alors les différents points du menu qui figurent dans le tableau. La ligne 410 déplace le curseur d'une position vers la droite et vers le bas, pour que les points du menu n'effacent pas le bord gauche du cadre du menu. La ligne 450 demande enfin à l'utilisateur d'entrer une valeur. Comme l'instruction PRINT de cette ligne est suivie d'un point-virgule, l'entrée avec INPUT sera attendue immédiatement après la parenthése (1-6).

Voilà donc comment nous avons créé notre menu. Il faudrait encore contrôler si la valeur entrée avec INPUT est correcte car nous ne pouvons pas avec INPUT sélectionner les touches à utiliser comme avec INKEY$.

Remarquons encore simplement que vous pouvez appeler le sous-programme de création de la ligne-titre de n'importe quel endroit de votre programme, avec l'instruction :

GOSUB 230

Notre but n'était pas ici de réaliser un programme complet mais uniquement de vous montrer la marche à suivre pour créer un menu. Cependant cela constituerait pour vous un excellent exercice si vous essayiez de complèter notre projet de programme. Le principe à suivre sera d'ailleurs le même que pour le programme de cours de calcul. Mais nous allons maintenant revenir plus en détail sur l'instruction INKEY$.

4.3.1 UTILISATION DE ROUTINES INKEY$ DANS LE MENU

Lorsque nous avons appris à utiliser l'instruction INKEY$, nous avions vu qu'elle présente l'inconvénient de ne pas permettre à l'utilisateur de voir ce qu'il entre dans l'ordinateur. Nous avons trouvé dans la dernière version du cours de calcul un moyen de remédier à cet inconvénient en faisant simplement sortir avec une instruction PRINT les caractères lus par A$=INKEY$.

Vous avez peut-être cependant vous-même remarqué que cette méthode est très "primitive" puisqu'elle ne permet pas à l'utilisateur de corriger son entrée et qu'elle le contraint à entrer exactement le nombre de caractères attendu. C'est ainsi que dans notre dernière version du cours de calcul, l'utilisateur doit entrer 054 pour 54 car le programme attend de toute façon trois chiffres.

C'est pourquoi il convient de reconnaître que l'instruction INPUT est préférable et tout à fait satisfaisante pour un usage privé. Mais si vous voulez réaliser un programme de niveau commercial, il faut utiliser l'instruction INKEY$ qui vous permet seule de protéger votre programme et l'utilisateur contre des erreurs de manipulation.

Nous allons donc développer une routine d'entrée avec l'instruction INKEY$. Vous pourrez intégrer cette routine sous forme d'un sous-programme dans vos propres programmes. La première ligne en est maintenant connue :

```
10 A$=INKEY$:IF A$="" THEN 10
```

Cette ligne vous permet de lire tous les caractères entrés à partir du clavier et de les affecter au fur et à mesure à A$. Dans notre application, nous n'autoriserons que les chiffres.

Il nous faut donc sélectionner les caractères à entrer et exclure toutes les autres touches :

```
10 A$=INKEY$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
```

L'instruction IF ... THEN exclut ici tous les caractères dont le code ASCII n'est pas compris entre 48 et 57 qui sont les codes de 0 et 9. Si un caractère incorrect est entré, il est donc ignoré et le programme saute à nouveau à la ligne d'entrée.
Maintenant, si nous ne voulons autoriser que des nombres comprenant au maximum un certain nombre de chiffres, il nous faut compter le nombre de caractères corrects entrés :

```
10 A$=INKEY$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Il nous a fallu introduire deux lignes supplémentaires: la ligne 30 qui augmente la valeur du compteur Z chaque fois qu'un caractère correct a été entré et la ligne 40 qui empêche que plus de 4 caractères ne soient pris en compte.

Il nous faut maintenant trouver un moyen pour indiquer à notre routine quand une entrée est terminée. Nous pouvons utiliser tout simplement pour ce faire la touche ENTER dont le code ASCII est 13. Il suffira de tester si ASC(A$) est égal à 13. Mais il faut faire attention à l'endroit où nous pourrons intégrer ce test. En effet si nous le plaçons après la ligne 20, la touche ENTER sera ignorée comme toutes les touches qui ne correspondent pas à des chiffres. Voici notre dispositif d'entrée complété :

```
10 A$=INKEY$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
```

```
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Il nous faut maintenant stocker dans une variable les différents caractères qui pourront être entrés dans ce dispositif. Il faut également que les caractères entrés soit sortis sur l'écran :

```
10 A$=INKEY$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
```

La ligne 60 sort les caractères à partir de l'emplacement actuel du curseur. Les différents caractères seront sortis l'un à la suite de l'autre grâce au point-virgule. Mais il faut ensuite que le programme retourne en ligne 10 après avoir sorti chaque caractère :

```
10 A$=INKEY$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
```

Il ne nous reste plus qu'à convertir la chaîne de caractère B$ en une valeur numérique et à affecter celle-ci à une variable numérique. Ceci doit être fait une fois que l'utilisateur a actionné la touche ENTER pour mettre fin à son entrée.

Comme nous voulons d'autre part utiliser cette routine sous la forme d'un sous-programme, nous ne devons pas oublier de réinitialiser à 0 le compteur Z chaque fois que la touche ENTER a été actionnée. Sinon l'utilisateur ne pourrait, lors du prochain appel de cette routine, entrer qu'un nombre de chiffres égal à 4 moins la valeur atteinte par Z lors de sa première utilisation.

Nous pourrions enfin prévoir de sortir en fin de routine la valeur numérique entrée par l'utilisateur, puis il serait temps de terminer notre routine par une instruction RETURN :

```
10 A$=INKEY$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
100 B=VAL(B$)
110 PRINT B
120 END
```

Nous avons ainsi réalisé une routine avec INKEY$ qui nous permet d'entrer et de sortir sur l'écran des nombres comportant jusqu'à 4 chiffres. Si vous voulez pouvoir entrer des nombres plus importants, il vous suffit de changer la valeur de Z en ligne 40.

Cette routine est plus pratique que celle que nous avions dans notre programme de cours de calcul. Il lui manque cependant une fonction importante, qui est d'ailleurs assez complexe à réaliser, à savoir la possibilité d'effacer les valeurs qu'on vient d'entrer. Voici une routine qui possède cette fonction :

```
10REMROUTINE INKEY$ AVEC FONCTION D'ANNULATION
20 A$=INKEY$:IF A$="" THEN 20
30 IF ASC(A$) = 13 THEN 130
40 IF ASC(A$) <> 127 THEN 70
50 IF LEN(B$) < 1 THEN 20
55 A$=CHR$(8)+CHR$(16)
60 B$=LEFT$(B$,LEN(B$)-1):Z=Z-1:GOTO 110
70 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 20
80 Z=Z+1
90 IF Z > 4 THEN 20
100 B$=B$+A$
110 PRINT A$;
120 GOTO 20
130 B=VAL(B$):Z=0
140 PRINT B
150 END
```

Voyons les lignes que nous avons ajoutées à la routine précédente. Tout d'abord la ligne 40 qui teste si la touche DEL a été actionnée. La valeur ASCII de cette touche est en effet 127. Si cette touche n'a pas été actionnée, le programme saute à la ligne 70. Si par contre la touche DEL a été actionnée, la ligne 50 teste si la variable alphanumérique B$ contient encore au moins un caractère. Si vous négligez ce dernier test, une tentative pour effacer un caractère d'une chaîne vide entraînerait l'interruption du programme avec le message d'erreur :

Improper argument in 60

Si donc la variable B$ est déjà vide, la touche DEL sera ignorée et le programme sautera à nouveau à la ligne 20.

L'effaçage d'un caractère est réalisé par la ligne 60 :

B$=LEFT$(B$,LEN(B$)-1)

qui raccourcit d'un caractère la variable B$.
En effet l'instruction LEFT$ génère ici une chaîne de caractères comprenant les X caractères les plus à gauche de B$ mais X est ici remplacé par LEN(B$)-1, c'est-à-dire la longueur antérieure de B$ moins 1. La nouvelle chaîne ainsi générée est à nouveau affectée à B$. Ce procédé est semblable à l'instruction très courante qui permet de diminuer la valeur d'une variable numérique :

A=A-1

La ligne 60 a donc pour effet de supprimer le caractère le plus à droite de B$, c'est-à-dire aussi le dernier entré. Le compteur du nombre de chiffres entrés Z est alors tout naturellement diminué de 1 de façon à ce que le caractère qui a été supprimé ne soit pas pris en compte par le test de la ligne 90 qui contrôle que l'utilisateur n'a pas entré plus de 4 chiffres.

Le caractère qui vient donc d'être supprimé de la variable qui mémorise votre entrée ainsi que de la variable-compteur Z qui compte combien de chiffres vous avez entrés, ce caractère se trouve cependant toujours sur l'écran. C'est pourquoi la dernière instruction de la ligne 60 est une instruction de saut en ligne 110 où la valeur actuelle de A$ est sortie sur l'écran par l'instruction PRINT A$;. Lorsque vous venez d'appuyer sur la touche DEL, A$ contient le code ASCII 127. Si ce code est sorti avec PRINT, cela produit un caractère graphique. Or ce qui nous intéresse, ce n'est pas de sortir un caractère graphique sur l'écran, mais bien de faire revenir le curseur d'une position en arrière et de supprimer le caractère qui se trouve à cet emplacement. Il nous faut pour arriver à ce résultat utiliser les codes de commande 8 et 16.

Entrez par exemple :

PRINT"CPC 465";CHR$(8);CHR$(16);"4"

Vous voyez que vous obtenez à l'écran la sortie suivante :

CPC 464

En effet le code de commande 8 ramène le curseur d'une position en arrière et le code de commande 16 efface le caractère qui se trouve dans l'emplacement actuel du curseur. Le "4" remplace alors immédiatement le caractère qui vient d'être ainsi supprimé.
C'est pourquoi nous avons remplacé en ligne 55 l'ancien contenu de A$ par les codes 8 et 16. La ligne 110 a alors pour effet de sortir ces deux codes de commande et de supprimer le dernier caractère sorti sur l'écran, chaque fois que vous appuyez sur la touche DEL.

Nous avons ainsi développé une routine INKEY$ qui est presque aussi simple à manier pour l'utilisateur qu'une routine INPUT mais qui présente en outre l'avantage de nous permettre de sélectionner les touches autorisées pour entrer une donnée.

Nous pouvons maintenant parachever notre routine en créant une sorte de curseur. Nous utiliserons comme curseur le trait graphique fin qui figure sur la touche 0. Ce caractère a le code ASCII 95. Nous utiliserons à nouveau le code de commande 8 pour déplacer le véritable curseur, qui reste bien sûr toujours invisible lorsqu'on utilise INKEY$, d'une position vers la gauche. En effet, chaque fois qu'un caractère sera entré, il sera sorti par PRINT A$; dans l'emplacement actuel du vrai curseur et il effacera donc "notre" curseur. Mais notre curseur (CHR$(95)) sera sorti à nouveau sur l'écran immédiatement après le dernier caractère entré (ligne 190).

Nous allons d'autre part rendre notre routine plus générale en autorisant, outre les chiffres, les lettres et les espaces. Il nous faut ajouter pour cela deux instructions IF ... THEN supplémentaires pour interdire les caractères dont les valeurs ASCII (de 58 à 64) sont entre celles des chiffres et celles des lettres. Voici donc notre routine :

```
10 REM ************************************
20 REM * POSITIONNEMENT DU CURSEUR *
30 REM ************************************
40 CLS
50 LOCATE 5,5
60 PRINT CHR$(95);CHR$(8);
70 REM *********************************
80 REM * ENTREE D'UN CARACTERE  *
90 REM *********************************
100 A$=INKEY$:IF A$="" THEN 100
110 IF ASC(A$) = 13 THEN 240
120 IF ASC(A$) = 32 THEN 180
130 IF ASC(A$) <> 127 THEN 150
140IFLEN(B$)>1THENB$=LEFT$(B$,LEN(B$)-1):A$=CHR$(16)+
  CHR$(8) + CHR$(16):GOTO 190
150 IF ASC(A$) < 48 THEN 100
160 IF ASC(A$) > 90 THEN 100
170 IF ASC(A$) > 57 AND ASC(A$) < 65 THEN 100
180 B$=B$+A$
190 PRINT A$;CHR$(95);CHR$(8);
200 GOTO 100
210 REM ****************************
220 REM * SORTIE DE LA CHAINE  *
230 REM ****************************
240 CLS
250 LOCATE 10,15
260 PRINT B$
```

Les lignes 40 et 50 vident l'écran et amènent le curseur en 5ème ligne 5ème colonne de l'écran. Le curseur que nous avons créé est alors sorti à cet emplacement (ligne 60) et le vrai curseur invisible est ramené d'une position en arrière (CHR$(8)). La routine d'entrée proprement dite commence en ligne 100. La ligne 120, qui a été ajoutée à la routine précédente, teste si le dernier caractère entré est un espace (CHR$(32)). La ligne 160 exclut les caractères dont le code ASCII est supérieur à celui du Z (90) et la ligne 170 ceux dont le code ASCII est compris entre celui du 9 (57) et celui du A (65) non-inclus.
Nous avons utilisé l'opérateur logique AND car les caractères ne doivent être exclus que s'ils remplissent deux conditions : en effet les caractères dont le code est supérieur à celui du 9 peuvent être des lettres. Ils ne doivent donc être exclus que si leur code est en même temps inférieur à celui de la première lettre, A.
Nous avons déjà expliqué l'effet de la ligne 190 mais nous allons essayer de le représenter symboliquement dans les lignes suivantes, pour qu'il soit encore plus clair. La position du curseur réel qui est invisible sur l'écran sera indiquée par une étoile.

Supposons que nous ayons entré la lettre A. L'effet de la ligne 190 sera alors le suivant :

A* après exécution de PRINT A$;

A_* après exécution de CHR$(95);

A* après exécution de CHR$(8);

Notre routine est maintenant très proche de l'instruction INPUT. Vous pouvez l'intégrer dans tous vos programmes. Il vous suffira de modifier les instructions IF ... THEN en fonction des touches que vous voulez autoriser ou interdire.

Vous voyez qu'on peut toujours reproduire une routine qui simule l'effet d'une instruction. L'intérêt est qu'on peut alors adapter beaucoup plus aisément cette routine à des besoins particuliers. Essayez par exemple de créer une routine BASIC qui puisse simuler dans un programme la fonction de l'instruction LOCATE.

4.3.2 TECHNIQUES DE FENETRES

Le CPC vous permet de définir jusqu'à 8 fenêtres différentes. Chaque fenêtre peut être appelée séparément pour une tâche différente. Les fenêtres peuvent entre autre permettre d'améliorer la présentation d'un menu. Vous pouvez par exemple faire afficher un message dans une fenêtre (Placer une cassette ! par exemple) sans endommager le menu proprement dit.

On pourrait par exemple définir la dernière ligne de l'écran comme une fenêtre dans laquelle s'afficheraient tous les messages du programme. L'instruction de définition d'une fenêtre est :

WINDOW #X,A,B,C,D

X est le numéro de la fenêtre, A et B correspondent aux colonnes limites de la fenêtre (gauche et droite) et C et D aux lignes limites supérieure et inférieure.

Nous pouvons donc définir ainsi une fenêtre ne comprenant que la dernière ligne de l'écran :

WINDOW #1,1,40,25,25

Après avoir entré cette instruction, vous ne constatez encore aucun changement visible.

Mais si vous entrez maintenant l'instruction suivante :

PRINT #1,"Ceci est la derniere ligne de l'ecran."

vous voyez que cette phrase a bien été sortie dans la fenêtre 1.

L'instruction PRINT # + le numéro de la fenêtre sortira donc toute expression dans la fenêtre voulue. Entrez maintenant :

CLS #1

Vous voyez que seul le contenu de la fenêtre 1 a été effacé.
Vous pouvez modifier la couleur d'une fenêtre avec l'instruction PAPER :

PAPER #1,3:CLS #1

remplira par exemple la fenêtre 1 de rouge. Vous pouvez donc ainsi souligner les contours d'une fenêtre à l'intérieur d'un menu. Si vous utilisez à nouveau l'instruction PRINT #1, vous verrez que la sortie dans la fenêtre se fera en jaune sur fond rouge.

Lorsque votre ordinateur est dans son état initial après allumage, il considère la totalité de l'écran comme la fenêtre de numéro 0. Si vous utilisez l'instruction CLS, vous effacez donc la totalité de l'écran, y compris votre fenêtre. Pour empêcher cela, vous pouvez réduire par exemple la fenêtre 0 d'une ligne avec l'instruction :

WINDOW #0,1,40,1,24

Si vous entrez maintenant CLS, le texte de la fenêtre 1 ne sera pas effacé.

La fenêtre de sortie standard est la fenêtre 0. Si vous voulez échanger les numéros de deux fenêtres, utilisez l'instruction :

WINDOW SWAP X,Y

Par exemple :

WINDOW SWAP 0,1

aura pour effet que la fenêtre de sortie standard 0 sera maintenant la dernière ligne de l'écran et que le reste de l'écran

constituera maintenant la fenêtre 1.

Vous avez maintenant en main tous les outils nécessaires pour réaliser de splendides menus que l'utilisateur aura plaisir à utiliser ! Souvenez-vous simplement qu'il ne faut jamais surcharger l'écran d'informations, sous peine de lasser l'utilisateur. Vous pourriez essayer maintenant d'appliquer vos nouvelles connaissances en réalisant le programme de tables mathématiques. Laissez libre cours à votre votre créativité dans la réalisation du menu en essayant d'intégrer les outils que nous venons d'étudier.

Les chapitres suivants seront une introduction aux algorithmes de tri et aux programmes de gestion de fichier.

4.4 PROCEDURES DE TRI

Le tri de données, qu'il s'agisse de données numériques ou de textes est une des tâches les plus classiques des programmeurs. Le critère de tri est naturellement la grandeur même des nombres lorsqu'il s'agit de données numériques et l'ordre alphabétique lorsqu'il s'agit de textes.

Pour trier des données on peut faire appel à quantité d'algorithmes bien connus qui se différencient les uns des autres d'une part par leur complexité et d'autre part par leur efficacité. Pour choisir le bon algorithme pour une application donnée il convient tout d'abord de tenir compte du nombre de données qu'il s'agit de trier. Si vous n'avez en effet qu'une dizaine ou une cinquantaine de données à trier, les procédures les plus simples suffiront en général aisément à la tâche. Par contre si la masse de données excède une centaine ou un millier, les procédures les plus simples se révèleront le plus souvent impraticables car le temps de tri risque de dépasser plusieurs heures. Nous allons vous présenter la méthode de tri la plus simple, qui est appelée bubble sort.

Avec bubble sort on compare deux éléments voisins qu'on échange si nécessaire. Une fois que tous les éléments d'un fichier ou d'un tableau ont été ainsi comparés, ils se trouvent dans l'ordre voulu.

Nous allons d'abord créer un tableau de 6 éléments que nous remplirons de valeurs aléatoires :

```
10 REM CREATION D'UN TABLEAU
20 DIM F(6)
30 FOR I=1 TO 6
40 A=INT(50*RND(0))+1
50 F(I)=A
60 NEXT I
```

Nous allons maintenant écrire notre procédure de tri. Bien sûr nous pourrions la réaliser en employant une boucle FOR ... NEXT mais nous pensons que l'algorithme utilisé sera plus évident si nous utilisons des instructions IF ... THEN. Une fois que vous aurez bien compris l'algorithme employé, vous pourrez bien sûr fort bien le mettre en oeuvre avec une boucle FOR ... NEXT. Voici donc notre routine de tri :

```
.
.
100 REM BUBBLE-SORT
110 Z=0
120 IF F(6) > F(5) THEN 140
130 F(0)=F(6):F(6)=F(5):F(5)=F(0):Z=1
140 IF F(5) > F(4) THEN 160
150 F(0)=F(5):F(5)=F(4):F(4)=F(0):Z=1
160 IF F(4) > F(3) THEN 180
170 F(0)=F(4):F(4)=F(3):F(3)=F(0):Z=1
180 IF F(3) > F(2) THEN 200
190 F(0)=F(3):F(3)=F(2):F(2)=F(0):Z=1
200 IF F(2) > F(1) THEN 220
210 F(0)=F(2):F(2)=F(1):F(1)=F(0):Z=1
220 IF Z=1 THEN 110
230 FOR I=1 TO 6
240 PRINT F(I)
250 NEXT I
260 END
```

Z est tout d'abord initialisé à 0 en ligne 110. Nous verrons plus loin pourquoi. La ligne 120 effectue la première comparaison: si F(6) est déjà supérieur à F(5), aucun échange n'est nécessaire et le programme saute directement à la seconde comparaison en ligne 140. Si par contre F(6) est inférieur à F(5), la ligne 130 échange entre eux les contenus de ces deux éléments.

C'est ici l'élément F(0) que nous utilisons comme variable de stockage provisoire pendant l'échange. Le même principe a été appliqué dans les lignes suivantes du programme.

Après chaque échange, la variable-compteur Z reçoit la valeur 1 alors que Z valait 0 en début de programme. Nous pouvons donc savoir grâce à la valeur de Z si au moins un échange de valeurs a été nécessaire: si Z vaut 0, cela signifie que les valeurs à trier étaient dans l'ordre voulu, si Z vaut 1 cela signifie qu'au moins deux valeurs n'étaient pas dans l'ordre voulu. Ce procédé qui consiste à utiliser une variable comme témoin d'une situation est très courant en informatique. Les variables-témoins sont appelées FLAGS, ce qui veut dire drapeaux. L'intérêt est ici que si aucune opération d'échange n'a été effectuée, donc si Z vaut 0, le programme sait qu'il peut mettre fin au tri.

Les dernières lignes de notre routine sortent le tableau trié sur l'écran. Mais si vous voulez observer le déroulement du tri, vous pouvez modifier ainsi les dernières lignes du programme :

```
.
.
220 FOR I=1 TO 6
240 PRINT F(I)
250 NEXT I
260 IF Z=1 THEN 110
270 END
```

Nous vous invitons maintenant à bien étudier cet algorithme de tri pour que vous n'ayez pas de mal à en aborder plus tard de plus complexes mais aussi plus performants. Cet algorithme peut être utilisé sans problème pour trier une masse d'environ 100 éléments.

4.5 LE PRINCIPE DE LA GESTION DE FICHIER

Un fichier (file en anglais) est un ensemble de données sauvegardées sur un moyen de stockage (cassette ou disquette) afin de pouvoir être rechargées dans l'ordinateur à tout moment. La gestion de fichier emploie toujours trois concepts fondamentaux : fichier, enregistrement et champ qui correspondent respectivement, le fichier au fichier de fiches en carton classiques, l'enregistrement à une fiche en carton classique et le champ à une rubrique de cette fiche en carton.

Suivant cette définition, les programmes sauvegardés sur cassette ou sur disquette constituent également des fichiers. Dans l'usage courant, on réserve cependant en général le terme de fichier aux fichiers de données par opposition aux fichiers de programmes.
Pour charger un programme d'une disquette dans votre CPC, entrez :

LOAD "nom du programme"

Le nom du programme entre guillemets doit comprendre au maximum 8 caractères. Vous pouvez entrer par exemple :

LOAD "EXEMPLE"

Le programme EXEMPLE sera alors chargé dans la mémoire de l'ordinateur.

Pour sauvegarder un programme sur disquette, entrez :

SAVE "nom du programme"

Par exemple :

SAVE "EXEMPLE"

Le programme EXEMPLE sera alors sauvegardé sur cassette.

Vous pouvez faire sortir sur l'écran le contenu d'une disquette avec l'instruction :

CAT

L'instruction SAVE vous permet de sauvegarder différents types de fichier. Il vous suffit d'indiquer le type de fichier à la suite de l'instruction SAVE et d'une virgule de séparation. Par exemple :

SAVE"Nom",A

sauvegardera le fichier Nom sous forme de fichier ASCII. Si vous remplacez le A par un P, vous pouvez sauvegarder un programme BASIC qui sera protégé et qui ne pourra donc plus être "listé" après avoir été chargé. Si vous utilisez un B comme type de fichier, votre fichier sera sauvegardé comme fichier binaire.

Le principe d'une gestion de fichier élémentaire est le suivant: vous entrez d'abord des données dans un tableau puis vous sauvegardez la totalité des données de ce tableau sur un moyen de stockage (disquette ou cassette). Un programme de gestion de fichier vous permet normalement de faire rechercher certaines données, de les modifier puis de sauvegarder à nouveau votre fichier ainsi modifié sur cassette ou sur disquette.

Si vous voulez transmettre des données au lecteur de disquette, vous devez d'abord ouvrir un fichier avec l'instruction OPENOUT :

OPENOUT "ADRESSES"

ouvrira par exemple un fichier en sortie sur disquette. Vous pourrez ensuite envoyer des données au lecteur de disquette, dans ce fichier.

C'est l'instruction PRINT #9 qui vous permet de transmettre des données au lecteur de disquette. 9 est le numéro de périphérique du lecteur de disquette. Si vous entrez :
PRINT #9,D$

le contenu de la variable alphanumérique D$ sera écrit sur la disquette. Pour mettre fin à la transmission de données, vous devez refermer le fichier avec l'instruction :
CLOSEOUT

Maintenant que nous savons comment écrire des données sur disquette, voyons comment lire des données à partir de la disquette :
Pour lire des données sur disquette, vous devez d'abord ouvrir un fichier cette fois avec l'instruction OPENIN :

OPENIN "ADRESSES"
ouvrira par exemple un fichier en lecture sur disquette. Vous pourrez ensuite lire les données sur disquette avec l'instruction INPUT #9. 9 est ici également le numéro de périphérique du lecteur de disquette. Si vous entrez :

INPUT #9,D$
des données écrites auparavant sur disquette seront lues et affectées à la variable D$. Pour mettre fin à la transmission de données, vous devez refermer le fichier avec l'instruction :

CLOSEIN
Vous savez maintenant tout ce qu'il est nécessaire de savoir pour comprendre notre prochain programme d'exemple. Il vous permet de choisir dans le menu si vous voulez entrer, sauvegarder, charger ou faire sortir sur l'écran des données. Si vous choisissez le point "entrée" de données, vous pouvez entrer quatre noms avec les prénoms correspondants.

Ce programme est volontairement simple et peu ambitieux pour que vous puissiez bien comprendre le principe de la gestion de fichier. Les connaissances en programmation que vous avez maintenant acquises devraient vous permettre désormais de développer vous-même un programme de gestion de fichier adapté à vos besoins.

```
10 DIM d$(4,2)
20 CLS
30 PRINT"Voulez vous"
40 PRINT
50 PRINT"Entrer des donnees ? (1)"
60 PRINT
70 PRINT"Sauvegarder des donnees ? (2)"
80 PRINT
90 PRINT"Charger des donnees ? (3)"
100 PRINT
110 PRINT"Sortir des donnees ? (4)"
120 PRINT
130 PRINT"Arreter ? (5)"
140 a$=INKEY$:IF a$="" THEN 140
150 ON VAL (a$) GOTO 190,290,400,500,600
160 REM **********************
170 REM * ENTREE DES DONNEES *
180 REM **********************
190 CLS
200 FOR i=1 TO 4
210 INPUT "Nom";d$(i,1)
220 INPUT "Adresse";d$(i,2)
230 CLS
240 NEXT i
250 GOTO 20
260 REM *************************
270 REM * SAUVEGARDE DES DONNES *
280 REM *************************
290 REM
300 OPENOUT "adresse"
310 FOR i=1 TO 4
320 FOR z=1 TO 2
330 PRINT #9,d$(i,z)
340 NEXT z,i
350 CLOSEOUT
360 GOTO 20
370 REM **************************
380 REM * CHARGEMENT DES DONNEES *
390 REM **************************
```

```
400 OPENIN "adresse"
410 FOR i=1 TO 4
420 FOR z=1 TO 2
430 INPUT #9,d$(i,z)
440 NEXT z,i
450 CLOSEIN
460 GOTO 20
470 REM ***********************
480 REM * SORTIE DES DONNEES *
490 REM ***********************
500 CLS
510 FOR i=1 TO 4
520 FOR z=1 TO 2
530 PRINT d$(i,z)
540 NEXT z,i
550 FOR i=1 TO 3000:NEXT i
560 GOTO 20
570 REM *******
580 REM * FIN *
590 REM *******
600 CLS
610 END
```

En conclusion de ce chapitre il nous reste à évoquer brièvement quatre instructions qui peuvent vous faciliter le travail avec les fichiers et le lecteur de disquette.

CHAIN

charge un programme dans l'ordinateur en supprimant le programme qui figure actuellement en mémoire. Les instructions :

CHAIN MERGE

et

MERGE

vous permettent de fusionner des programmes entre eux. Il faut cependant veiller à ce que les numéros de toutes les lignes d'un programme soient différentes des numéros de lignes existant dans l'autre programme.
L'existence de ces instructions sur votre CPC vous permet de constituer une bibliothèque de programmes. Vous pouvez en effet écrire puis sauvegarder certaines parties de programmes dont vous pensez que vous aurez souvent à vous en servir. Si vous avez ensuite besoin d'une partie de programme que vous avez ainsi sauvegardée, vous pourrez la fusionner avec un programme que vous êtes en train d'écrire grâce aux instructions MERGE et CHAIN MERGE.

La fonction EOF enfin vous permet de tester pendant le déroulement d'un programme si la fin d'un fichier a déjà été atteinte (EOF = End Of File). Si c'est le cas, la variable système EOF vaudra -1, si la fin du fichier n'a pas encore été atteinte, elle vaudra 0.

5. MUSIQUE ET GRAPHISME

Ce chapitre a pour but de vous présenter brièvement les instructions du CPC qui vous permettent de gérer les possibilités sonores et graphiques de votre ordinateur.

5.1 LA MUSIQUE

L'instruction SOUND est l'instruction de base pour produire des sons sur votre CPC. Elle a la syntaxe suivante :

SOUND A,B,C,D,E,F,G

Les différents paramètres ont la signification suivante :

A - état des canaux (valeur entre 1 et 255)
B - période, influe sur la fréquence (valeur entre 0 et 4095).
La fréquence = 125000/période
C - durée de la note (valeur entre -32768 et +32767). L'unité est le centiéme de seconde
D - volume (entre 0 et 15)
E - enveloppe du volume (entre 0 et 15)
F - enveloppe de la note (entre 0 et 15)
G - bruit (entre 0 et 15)

SOUND 1,284,100,12,0,0,0

Cette instruction produira par exemple le LA du diapason (fréquence 440 Hertz). La durée de la note sera d'une seconde et le volume 12.

L'instruction pour créer une enveloppe de volume est :

ENV

qui indique comment le volume d'un son devra croître puis retomber. L'instruction :

ENT

vous permet d'influer sur la hauteur même de la note, de façon à produire un vibrato. La fonction :

PRINT SQ(X)

vous permet d'interroger le canal d'états. X peut prendre les valeurs 1, 2 ou 4. L'instruction :

RELEASE X

supprime les points d'arrêt qui avaient été posés avec l'instruction SOUND. X peut prendre n'importe quelle valeur entre 1 et 7.

5.2 GRAPHISME

Le CPC possède trois modes graphiques. Après allumage, votre ordinateur se trouve en mode 1, ce qui correspond à une résolution graphique de 320 points sur 200. Dans ce mode vous pouvez utiliser 4 couleurs différentes à la fois. Le nombre de colonnes est de 40.

L'instruction :

MODE X

vous permet de sélectionner un des trois modes graphiques :

MODE 0

MODE 1

et

MODE 2

En mode 0 la résolution graphique n'est plus que de 160 points sur 200 et il n'y a plus que 20 colonnes. Mais vous pouvez utiliser simultanément 16 couleurs différentes. Le mode 2 réalise la meilleure résolution graphique: 640 points sur 200 et 80 colonnes. Mais vous ne pouvez plus utiliser que 2 couleurs.

Voici maintenant les instructions de programmation graphique :

BORDER X

change la couleur du cadre de l'écran. X doit être compris entre 0 et 31. Si vous entrez deux paramètres, la couleur du bord de l'écran clignotera entre deux couleurs.

SPEED INK X,Y

vous permet d'indiquer la vitesse de clignotement entre deux couleurs. Le délai entre deux clignotements doit être indiqué en cinquantièmes de seconde.

L'instruction :

CLG numéro de crayon couleur

remplit l'écran graphique de la couleur affectée au crayon de couleur indiqué.

DRAW X,Y

trace une ligne de l'emplacement actuel du curseur graphique à l'emplacement indiqué de manière absolue par les coordonnées X et Y.

DRAWR X,Y

trace une ligne de l'emplacement actuel du curseur graphique à l'emplacement indiqué relativement à l'emplacement du curseur par les coordonnées X et Y.

PLOT X,Y

allume un point de coordonnées absolues X et Y.

PLOTR X,Y

allume un point de coordonnées relatives X et Y.

MOVE X,Y

déplace le curseur graphique vers le point de coordonnées absolues X et Y.

MOVER X,Y

déplace le curseur graphique vers le point de coordonnées relatives X et Y.

XPOS

et

YPOS

fournissent les coordonnées du curseur graphique. Les valeurs de ces variables système peuvent être sorties avec PRINT ou affectées à une variable.

ORIGIN X,Y

modifie l'origine par rapport à laquelle sont indiquées les coordonnées absolues.

TAG

vous permet de faire sortir avec PRINT vos textes dans l'emplacement du curseur graphique.

TAGOFF

rétablit la sortie avec PRINT des textes dans l'emplacement du curseur normal de texte.

INK

vous permet de déterminer quelles couleurs doivent être utilisées.

Par exemple :

INK 1,3

affecte la couleur rouge au registre 1. Si vous entrez maintenant :

PEN 1

la couleur d'écriture sera maintenant celle du registre 1, donc le rouge.

PAPER

vous permet de déterminer de la même façon la couleur du fond de l'écran.

PRINT TEST(250,150)

vous permet de savoir par exemple avec quelle couleur a été dessiné le point de coordonnées absolues 250 et 150.

TESTR (X,Y)

est l'instruction correspondante pour le point de coordonnées relatives X et Y.

Instructions spécifiques au CPC 664 et 6128

FILL X

Remplit une zone à l'écran, délimitée par des lignes tracées soit avec l'encre graphique courante, soit avec l'encre qui sera utilisée pour le remplissage, avec l'encre définie par X.

MASK X,Y

X permet de définir le masque utilisé par "DRAW". X est compris entre 0 et 255. L'image binaire de X sera représenté par un pixel éteint pour un bit à 0 et un pixel allumé pour un bit à 1. La valeur donnée à Y détermine si le premier point de la ligne doit être affiché (Y = 1) ou s'il ne doit pas l'être (Y = 0).

FRAME

Synchronise l'écriture des graphiques sur l'écran avec l'affichage sur le moniteur. Cela produit un mouvement régulier des graphiques qui se déplacent.

CURSOR X

Permet de faire apparaître ou non, le curseur pendant le déroulement d'un programme.
X = 1 curseur visible,
X = 2 curseur invisible.

GRAPHICS PEN X,Y

X séléctionne l'encre utilisée par "DRAW", "PLOT" et l'affichage graphique des caractères avec "TAG". Si Y est égal à 0, la couleur de fond des caractères sera celle définie par "GRAPHICS PAPER" et celle du papier si Y a pour valeur 1.

GRAPHICS PAPER X

Détermine l'encre de fond pour l'affichage graphique de caractères avec "TAG".

ANNEXE 1

CODES ASCII

A) CODES DE COMMANDE

Déc.	Hex.	
0	0	aucun effet
1	1	sortie d'un symbole déterminé par le second paramètre
2	2	déconnecter curseur de texte
3	3	rétablir curseur de texte
4	4	avec un second paramètre, même fonction que MODE
5	5	sortie d'un symbole dans l'emplacement du curseur graphique
6	6	active l'écran texte
7	7	bip
8	8	curseur en arrière d'une case
9	9	curseur en avant d'une case
10	A	curseur une ligne vers le bas
11	B	curseur une ligne vers le haut
12	C	comme CLS
13	D	CARRIAGE RETURN (ENTER)
14	E	avec un second paramètre, même fonction que PAPER.
15	F	avec un second paramètre, même fonction que PEN.

16	10	supprimer caractère sous le curseur
17	11	supprimer ligne jusqu'au curseur
18	12	supprimer ligne à partir du curseur
19	13	vider l'écran jusqu'au curseur
20	14	vider l'écran à partir du curseur
21	15	déconnecter l'écran-texte
22	16	transparent oui/non (1/0)
23	17	fixer mode de crayon couleur pour graphisme
24	18	REVERS oui/non
25	19	avec 9 paramètres, même fonction que SYMBOL
26	1A	avec paramètres, même fonction que WINDOW
27	1B	aucun effet
28	1C	comme INK
29	1D	comme BORDER
30	1E	placer le curseur dans l'angle supérieur gauche de l'écran
31	1F	comme LOCATE
32	20	espace

B) CODES NORMAUX

Déc.	Hex.	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;

Déc.	Hex.		Déc.	Hex.	
60	3C	<	81	51	Q
61	3D	=	82	52	R
62	3E	>	83	53	S
63	3F	?	84	54	T
64	40	@	85	55	U
65	41	A	86	56	V
66	42	B	87	57	W
67	43	C	88	58	X
68	44	D	89	59	Y
69	45	E	90	5A	Z
70	46	F	91	5B	[
71	47	G	92	5C	\
72	48	H	93	5D	]
73	49	I	94	5E	↑
74	4A	J	95	5F	_
75	4B	K	96	60	`
76	4C	L	97	61	a
77	4D	M			
78	4E	N			
79	4F	O			
80	50	P			

Déc.	Hex.		Déc.	Hex.	
98	62	b	113	71	q
99	63	c	114	72	r
100	64	d	115	73	s
101	65	e	116	74	t
102	66	f	117	75	u
103	67	g	118	76	v
104	68	h	119	77	w
105	69	i	120	78	x
106	6A	j	121	79	y
107	6B	k	122	7A	z
108	6C	l	123	7B	{
109	6D	m	124	7C	|
110	6E	n	125	7D	}
111	6F	o	126	7E	~
112	70	p	127	7F	▒

ANNEXE 2

MOTS RESERVES

ABS, AFTER, AND, ASC, ATN, AUTO

BIN$, BORDER

CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN, CLOSEOUT, CLS, CONT, COS, CREAL

DATA DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE, DI, DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR, EVERY, EXP

FIX, FN, FOR, FRE

GOSUB, GOTO

HEX$, HIMEM

IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT

JOY

KEY

LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10,
LOWER$

MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT

ON, ON BREAK, ON ERROR GOTO, ON SQ, OPENIN, OPENOUT, OR,

MOTS RESERVES

ORIGIN, OUT

PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT

RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN

SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR, STEP, STOP, STR$, STRING$, SWAP, SYMBOL

TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON

UNT, UPPER$, USING

VAL, VPOS

WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE

XOR, XPOS

YPOS

ZONE

POUR LES CPC 664 ET 6128

FILL - MASK - FRAME - CURSOR - GRAPHICS

ANNEXE 3

CODE CLAVIER

Touche	Code	Touche	Code
ESPACE	47	8	40
ESC	66	9	33
TAB	68	0	32
CAPS LOCK	70	-	25
SHIFT	21	A	69
DEL	79	B	54
ENTER	18	C	62
CTRL	23	D	61
CLR	16	E	58
↑	24	F	53
1	64	G	52
2	65	H	44
3	57	I	35
4	56	J	45
5	49	K	37
6	48		
7	41		

Touche	Code
L	36
M	38
N	46
O	34
P	27
Q	67
R	50
S	60
T	51
U	42
V	55
W	59
X	63
Y	43
Z	71
,	39
.	31
/	30
\	22
:	29
;	28
[	17
]	19
@	26

MICRO APPLICATION MICRO APPLICATION

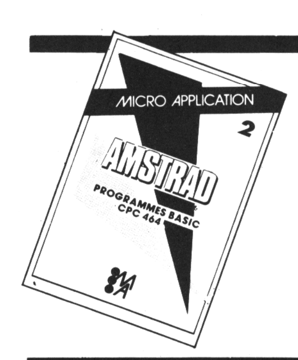
LES LIVRES AMSTRAD

TRUCS ET ASTUCES POUR L'AMSTRAD CPC (Tome 1)

C'est le livre que tout utilisateur d'un CPC doit posséder. De nombreux domaines sont couverts (graphismes, fenêtres, langage machine) et des super programmes sont inclus dans ce best-seller (gestion de fichiers, éditeur de texte et de sons...).

Réf. : ML112
Prix : 149 FF





PROGRAMMES BASIC POUR LE CPC 464

ALIMENTEZ VOTRE CPC 464

Ce livre contient de super programmes, notamment un désassembleur, un éditeur graphique, un éditeur de texte... Tous les programmes sont prêts à être tapés et abondamment commentés.

Réf. : ML119
Prix : 129 FF

LE BASIC AU BOUT DES DOIGTS CPC 464

Ce livre est une introduction complète et didactique au BASIC du micro-ordinateur AMSTRAD CPC 464. Il permet d'apprendre rapidement et facilement la programmation (ins-tructions BASIC, analyses des problèmes, algorythmes complexes...)
Principaux thèmes abordés :
- Les bases de la programmation
- Bit, Octet, ASCII
- Instructions du BASIC
- Organigrammes
- Les fenêtres
- Programmes BASIC plus poussés
- Le programme et menus.

Comprenant de nombreux exemples, ce livre vous assure un apprentissage simple et efficace du BASIC CPC 464.

Réf. : ML118
Prix : 149 FF



MICRO APPLICATION MICRO APPLICATION

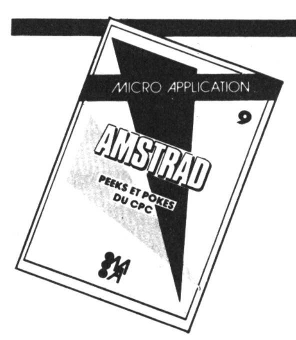
MICRO APPLICATION MICRO APPLICATION

GRAPHISMES ET SONS DU CPC

L'AMSTRAD CPC dispose de capacités graphiques et sonores exceptionnelles. Ce livre en montre l'utilisation à l'aide de nombreux programmes utilitaires.
Contenu :
- base de programmation graphique
- éditeur de police de caractères
- "sprites", "shapes", et chaînes
- représentations multi-couleurs.
- calcul des coordonnées
- rotations, mouvements
- représentations graphiques de fonctions en 3D
- D.A.O. (dessin assisté par ordinateur)
- synthétiseur
- mini-orgue
- enveloppes de son, et beaucoup d'autres choses...

Réf. : ML124
Prix : 129 FF





PEEKS ET POKES DU CPC (Tome 9)

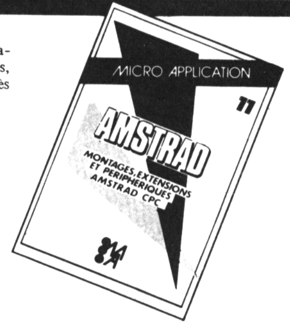
Comment exploiter à fond son CPC à partir du BASIC ? C'est ce que vous révèle ce livre avec tout ce qu'il faut savoir sur les peeks, pokes et autres call... Vous saurez aussi comment protéger la mémoire, calculer en binaire... et tout cela très facilement. Un passage, assuré et sans douleur du BASIC au puissant LANGAGE MACHINE.

Réf. : ML126
Prix : 99 FF

MONTAGES, EXTENSIONS ET PERIPHERIQUES AMSTRAD CPC (Tome 11)

Pour tous les amateurs d'éléctronique ce livre montre ce que l'on peut réaliser avec un CPC. De nombreux schémas et exemples illustrent les thèmes et applications abordés comme les interfaces, programmateur d'EPROM... Un très beau livre de 450 pages.

Réf. : ML131
Prix : 199 FF



MICRO APPLICATION MICRO APPLICATION

MICRO APPLICATION MICRO APPLICATION

LE LIVRE DU CP/M AMSTRAD (Tome 12)

Ce livre vous permettra d'utiliser CP/M sur les CPC 464, 664 et 6128 sans aucune difficulté. Vous y trouverez de nombreuses explications et les différents exemples vous assureront une maitrise parfaite de ce très puissant système d'exploitation qu'est CP/M. (300 pages).

Réf. : ML128
Prix : 149 FF





DES IDEES POUR LES CPC (Tome 13)

Vous n'avez pas d'idées pour utiliser votre CPC (464, 664, 6128) ? Ce livre va vous en donner !. Vous trouverez de très nombreux programmes BASIC couvrant des sujets très variés qui transformeront votre CPC en un bon petit génie. De plus les programmes vous permettront d'appronfondir vos connaissances en programmation.
(250 pages).

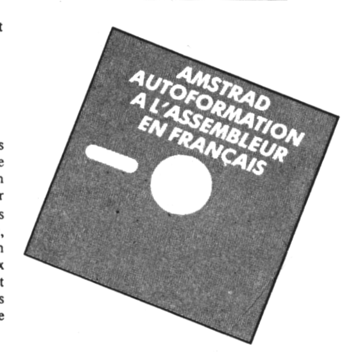
Réf. : ML132
Prix : 129 FF

AMSTRAD AUTOFORMATION A L'ASSEMBLEUR EN FRANCAIS

Contient un livre et un logiciel.
LE LIVRE :
Cet ouvrage introduit le débutant à la programmation du Z80 grâce à la méthode du DR WATSON qui selon les critiques vaut son pesant d'or ! Aucune connaissance préalable n'est requise et le but du livre est d'assurer au novice un succès total. A la fin du livre les instructions du Z80 sont expliquées en détail. De nombreux exemples illustrent les différentes étapes du cours alors que des exercices (les solutions sont fournies) testent la compréhension.
LE LOGICIEL : Un assembleur Z80 complet est livré sur cassette et comprend :
- Etiquettes Symboliques
- Directives d'Assemblage
- Chargement/Sauvegarde
- Copie Ecran
- INSERT / DELET.

L'assembleur permet d'écrire des programmes facilement en langage d'assemblage puis les transforme en code machine (langage machine). Pour vous aider à comprendre les rotations mathématiques utilisées, une démonstration de l'utilisation des nombres binaires et hexadécimaux est fournie. Un programme utilisant les commandes graphiques additionnelles décrites dans le livre est également fourni.

Réf. : ML126
Prix : 195 FF K 7 - **295 FF** - disquette



MICRO APPLICATION MICRO APPLICATION


MICRO
INFO
T23
XB2
ATARI
COMMODORE
AMSTRAD
80 pages de trucs et astuces, programmes, dossiers pour Amstrad CPC, Commodore Atari ST ...
20 f

MICRO APPLICATION vous présente MICRO INFO, nouveau journal avec des dossiers, des bidouilles, des trucs et astuces, des nouveautés, des programmes et plein de rubriques sympas! (88 pages)
Chaque numéro traite principalement de 3 matériels:

AMSTRAD - COMMODORE - ATARI

carte d'abonnement

Je désire m'abonner à MICRO INFO

- ☐ **Le numéro 1 : 15 F + 5 F pour frais d'envoi**
- ☐ **Le numéro 2 : 20 F + 5 F pour frais d'envoi**
- ☐ **Les numéros 1 et 2 : 35 F + 5 F pour frais d'envoi**
- ☐ **Je choisis de m'abonner pour 4 numéros au prix de 70F**

Je règle par ☐ chèque
☐ mandat
☐ CCP

Nom : ________________ Prénom : ______________
Adresse : ______________________________
Code postal : ________________ date et signature :

Veuillez nous retourner cette carte sous pli ainsi que votre règlement à l'adresse suivante :

MICRO APPLICATION
13 rue Sainte Cécile 75009 PARIS

Achevé d'imprimer en février 1986
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : février 1986
Numéro d'imprimeur : 602028

IMPRIMÉ EN FRANCE

[redacted]

Ce livre est une introduction complète et didactique au BASIC du micro-ordinateur AMSTRAD CPC 464, 664 et 6128.

Il permet d'apprendre rapidement et facilement la programmation (instructions BASIC, analyses des problèmes, algorithmes complexes...).

Principaux thèmes abordés :

- **Les bases de la programmation ;**
- **Bit, Octet, ASCII ;**
- **Instructions du BASIC ;**
- **Organigrammes ;**
- **Les fenêtres ;**
- **Programmes BASIC plus poussés ;**
- **Sous-programmes et menus.**

Comprenant de nombreux exemples, ce livre vous assure un apprentissage simple et efficace du BASIC de l'AMSTRAD.

ISBN : 2-86899-006-1
PRIX : 149 FF

Réf. : ML 118



AMSTRAD
LE BASIC AU BOUT DES DOIGTS
CPC 464, 664 et 6128
3
MA



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/